CoLogNetWS: the CoLogNet Web-site

Development of a Web Site using Persistent Predicates, PiLLoW and CGI
Development of a database multinode distributed update

J.A. Navas and D. Cabeza and M. Hermenegildo
clip@dia.fi.upm.es
http://www.clip.dia.fi.upm.es/
The CLIP Group
Facultad de Informática
Universidad Politécnica de Madrid

Summary

`CoLogNetWS` is a *Web-site* on Computational Logic systems, environments, and implementation technology.

`CoLogNetWS` provides at the same time:

- A simple WWW interface which allows the users to access/modify the data stored in its database.
- An automatic data exchange between `CoLogNetWS` and the rest of Web-sites, in order to keep their databases up-to-date.

This document constitutes an *internals* manual, providing information on how the different internal parts of `CoLogNetWS` are connected.

1 Introduction

`CoLogNetWS` is a Web-site on computational logic systems, environments, and implementation technology, which belongs to CoLogNet[1]. `CoLogNetWS` provides a simple *WWW interface* which allows users to access/modify data stored in its database. Besides, `CoLogNetWS` allows to automatically *data exchange* between `CoLogNetWS` and the rest of Web-sites, in order to keep their databases up-to-date.

As mentioned above, `CoLogNetWS` provides information about Computational Logic, specifically about researchers, research groups, research areas, research projects, systems and environments, as well as events related to this research topic.

CoLogNetWS is implemented in The Ciao Program Development System. The database is built according to the definition of file-based *persistent predicates*, implemented in the library `persdb` [GCH98] of Ciao and is directly maintained by `CoLogNetWS`. Regardless to this the rest of CoLogNet Web-sites are free to implement their databases according to other technologies.

As will be shown, the information can be accessed by two different ways. For each way is defined an interface: *internal interface* and *external interface*.

Internal interface is a simple WWW interface based on *HTML* forms. This interface is implemented by means of the library `PiLLoW` [CH01] of Ciao. `PiLLoW` provides facilities for accessing documents and code on the WWW; parsing, manipulating and generating HTML and *XML* structured documents and data; producing HTML forms; writing forms handlers and *CGI*-scripts, and processing HTML/ XML *templates*.

The internal interface is divided into two sub-interfaces: *user interface* and *manager interface*. Through the first interface, users can carry out several operations such as *View*, *Insert*, *Modify* or *Delete* an item corresponding to an entity. Some of them, such as *View* or *Insert* don't require *authentication*. However, other operations like *Modify* and *Delete* require authentication. The manager interface allows a complete access to data, avoiding that authentication. Besides, offers extra functionalities, depicted in this document.

The external interface allows communication between `CoLogNetWS` and the rest of Web-sites enabling automatic database update. The communication protocol among the different Web-sites is based on the *HTTP protocol*, and the information exchange is encoded in XML format.

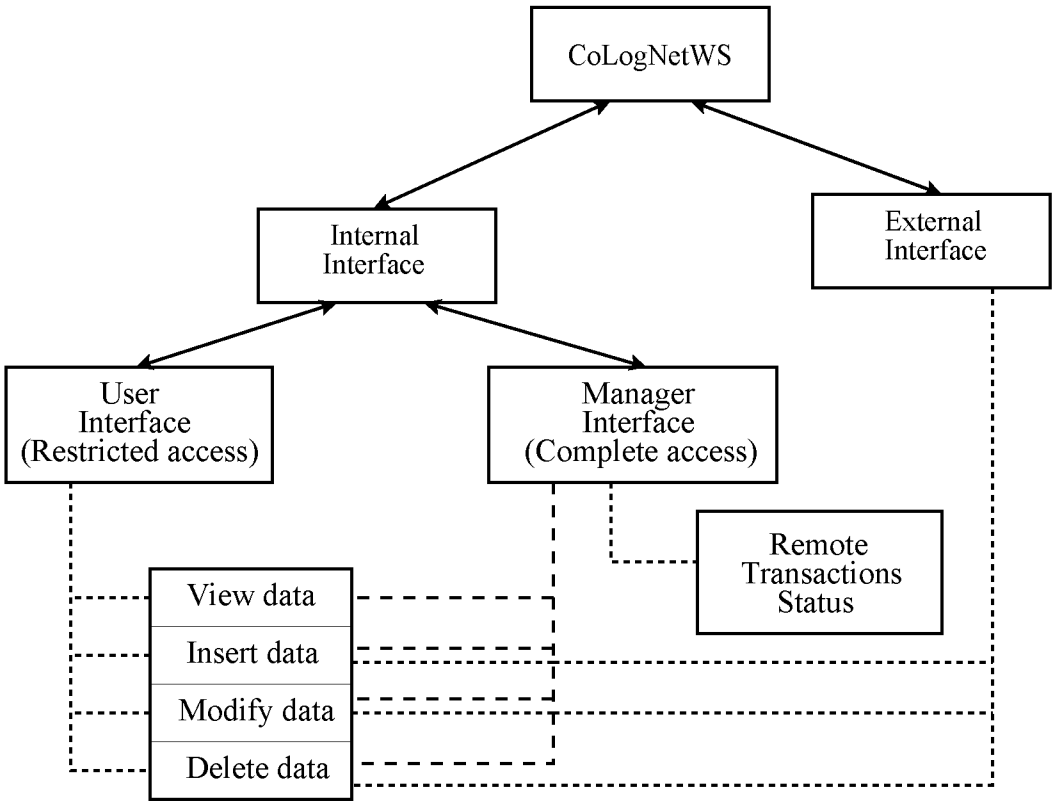
1.1 Structure of the Web-site

`CoLogNetWS` is designed to be user-contributed, so that it can be gradually completed and updated. The `CoLogNetWS` data is divided into: groups, people, projects, software, areas and events.

- The Groups section stores information about research groups working in an area.
- The People section consists of the information about research groups/individuals working in the area.
- The Projects section lists research projects related to this research topic.
- The Software section provides information about systems and environments for Computational Logic, as well as a database of application libraries.
- The Events section lists upcoming conferences, workshops and another events related to this research topic.
- The Areas section lists the main areas Computational Logic and Logical Applications is divided into.

[1] CoLogNet is the EU-funded Network of Excellence in Computational Logic. CoLogNet aims to help unify and integrate the separate sub-communities in the area of Computational Logic and Logical Application

As we can see in the next figure, `CoLogNetWS` is composed by several views. Each view offers the appropriate interface for a given service, some of them allowed to the manager administrator only.

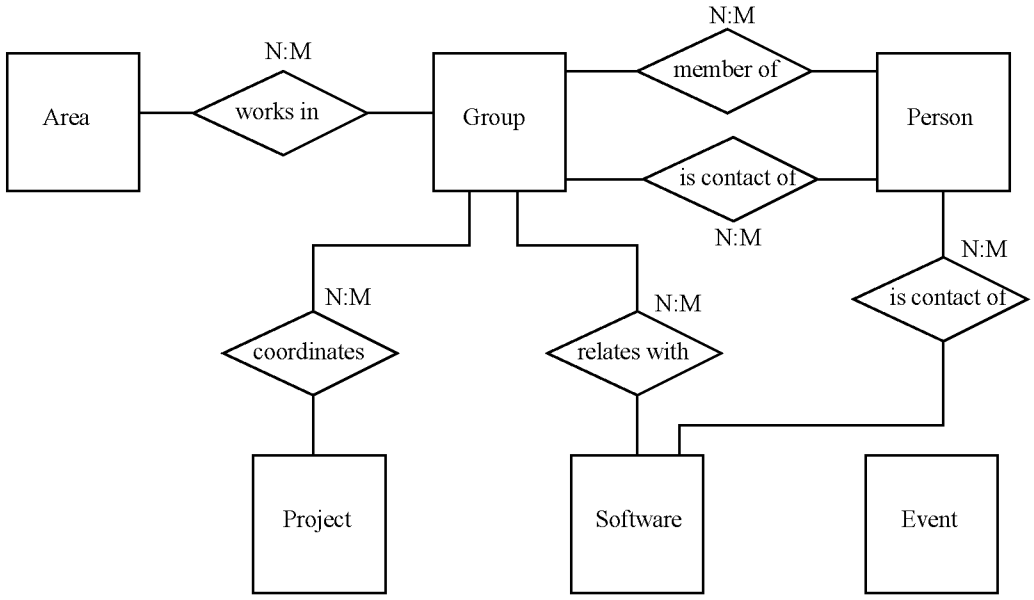


1.2 Data Storage

`CoLogNetWS` data is local, but the design allows to easily modify it to be distributed. The data storage is based on the concept of persistent predicates. The idea is that whatever changes might have been done on one of these special kind of dynamic predicates survive across executions. This way, no matter whether the system stops, or even crashes, we will always have a consistent state of the predicate, which is quite an useful thing to implement a database. If the system is halted and restarted, the predicate the new Prolog process sees is right in the state it was in the very moment the old process was halted, provided that no changes to the external storage were done in the meantime by other processes or the user himself.

So, all `CoLogNetWS` data storage related to the entities mentioned in the previous section is based on persistent predicates. Persistent predicates are also used to manage some information the system needs to maintain across CGI executions, such as security information.

The database design is based on the *Relational model*. The next figure shows the database design according to *Cheng* notation.



1.3 Internal interface

As already mentioned, the internal interface is composed by two sub-interfaces: user interface and manager interface.

Both user interface and manager interface can basically carry out four operations: *View*, *Insert*, *Modify* and *Delete* an item corresponding to an entity, meaning by entity a group, person, project, software, area or event.

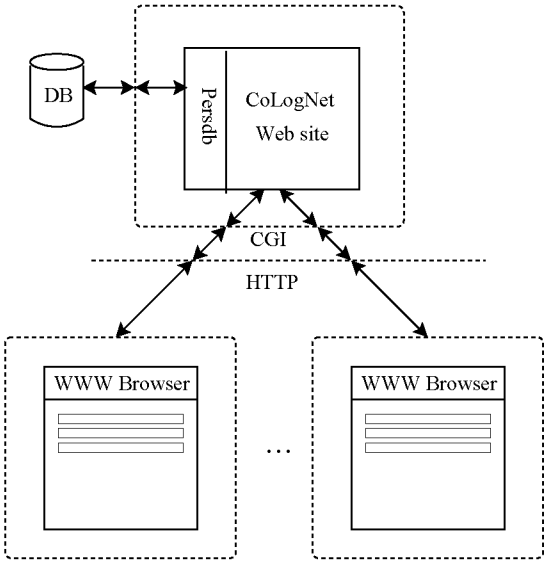
- *View* an item shows all data about an item corresponding to an entity. This information depends on each entity.
- *Insert* an item allows to store the new item data in the `CoLogNetWS` database.
- *Modify* an item allows to change the item data which is already stored in the `CoLogNetWS` database.
- *Delete* an item allows to delete the item data.

Regarding the *graphic interface*, it is implemented using HTML templates. These templates allow the layout of the output pages to be easily configurable and independent of the application logic level.

Additionally, an important operation that users can carry out both through the user interface and the manager interface, is the interaction with the system by means of XML files. An item or a group of items corresponding to an entity or entities can be inserted, modified or deleted using an XML file, instead of using an bothersome HTML forms. That is if you want to insert a new group and another related data, such as memberships, projects, software and areas, across an HTML form, you would have to fill multiple HTML forms, one for each item.

`CoLogNetWS` must to have two different views: normal user and administrator, due to the administrator has more privileges than the normal user. In order to fulfill this requirement, user interface and manager interface are offered by the application. The next figure shows the

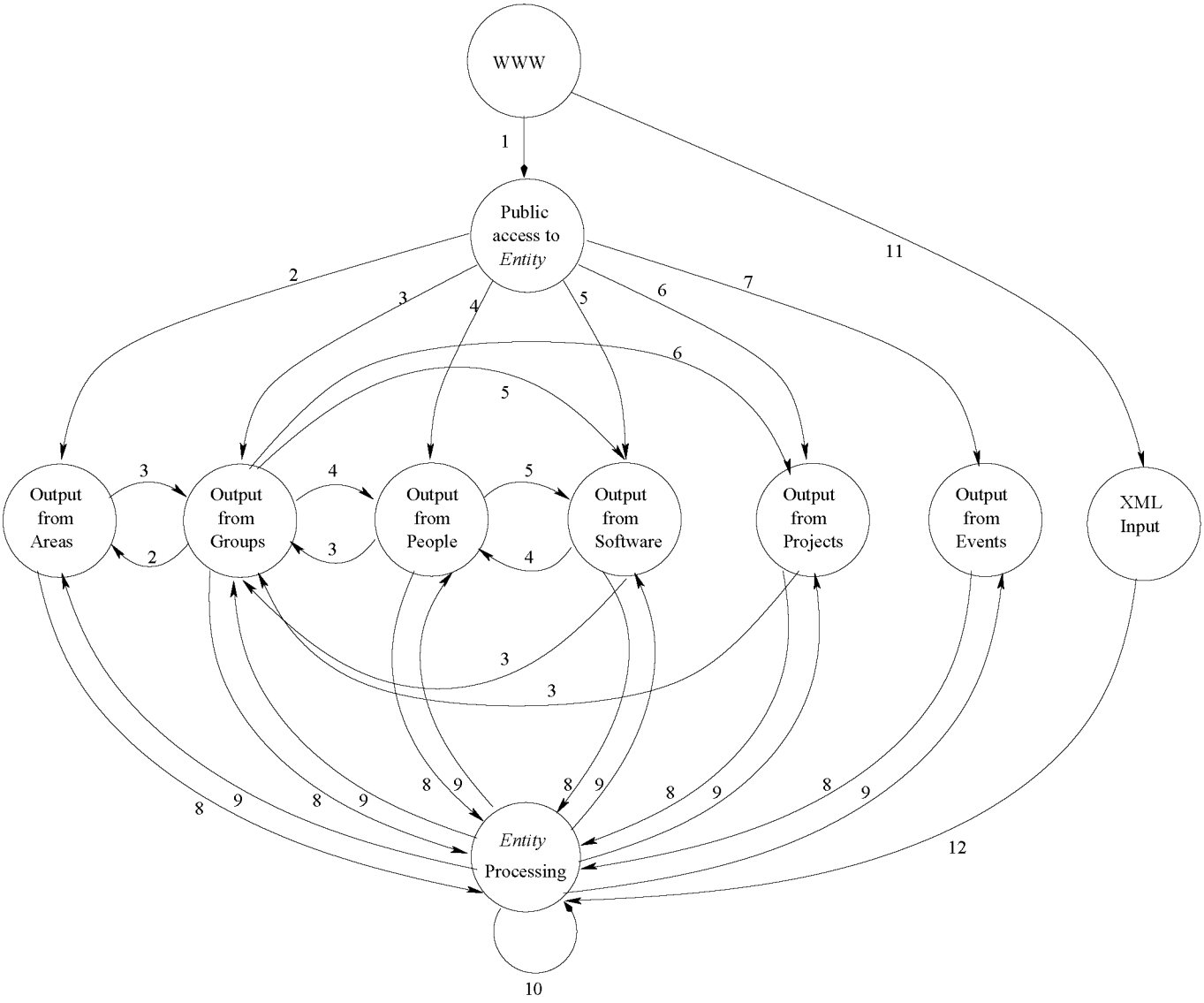
interaction between `CoLogNetWS` and an user, who can be either a normal user or a manager user.



As seen in the figure, the services of `CoLogNetWS` are provided through the WWW. These services basically consist of an interface to query and operate the database and, in the case of the system administrator, manage the system. We describe the particular features of each sub-interface.

1.3.1 User interface

This interface allows users to *view*, *insert*, *modify* or *delete* an entity. If the user is not an authorized client, the operations allowed will only be *view* and *insert*. In the next figure, it can be seen the interaction among the different processes belonging to the user internal interface:



- *1.* Access to user internal interface.
- *2.* View areas.
- *3.* View groups.
- *4.* View people
- *5.* View software.
- *6.* View projects.
- *7.* View events.
- *8.* Insert *"Entity"* / Modify *"Entity"* / Delete *"Entity"* / View related *"Entity"*.
- *9.* Response to insert *"Entity"* / Response to modify *"Entity"* / Response to delete *"Entity"* / Response to view related *"Entity"*.

- *10.* Ask password / Confirm Password / Insert Form / Modify Form / Insert / Modify.
- *11.* XML file input.
- *12.* XML file processing.

"Entity" can be: *area*, *group*, *person*, *project*, *software* or *event*.

The steps an user can take to access `CoLogNetWS` in user or public mode are:

First of all, the user accesses the system across WWW interface and the *"Public access to Entity"* process is executed for the corresponding entity. This process includes security operations, by granting the user restricted access to the system.

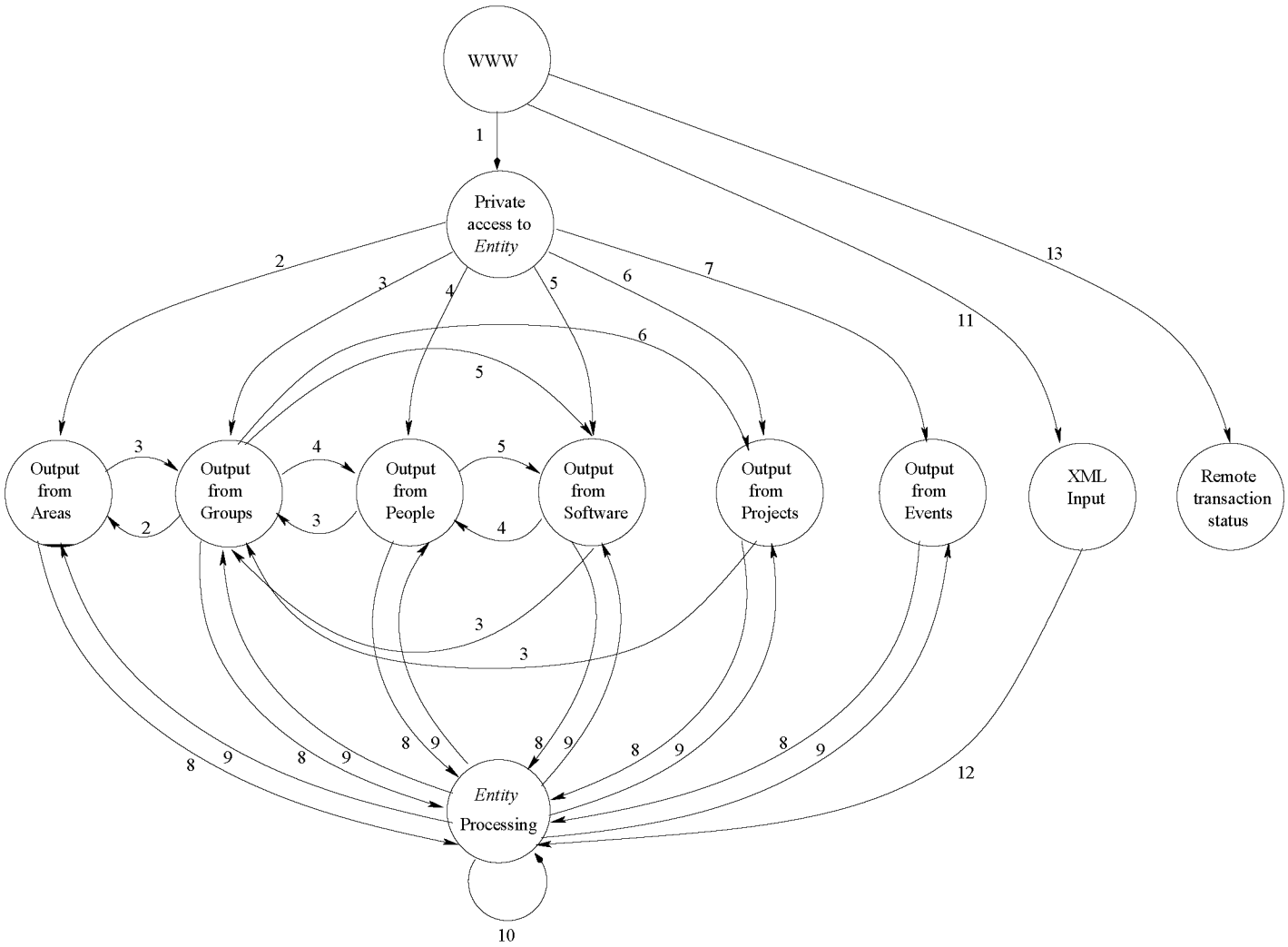
The next step is to show or view the content of all entities by means of a set of processes (*"Output from Areas"*, *"Output from Groups"*, *"Output from People"*, *"Output from Projects"*, *"Output from Software"*, *"Output from Events"*). Note there are interactions among these processes. These are based on the fact that users can see the content of an entity and access others entities from that one. For example, if we are seeing group data, we can directly access the people belonging to it. The *"Entity Processing"* process allows executing operations such as *insert*, *modify* and *delete*. Operations such as *modify* and *delete* need the user to get authenticated previously by means of a password. This password has to be the same one that the user inserted in the item registration or if the user didn't insert a password, it will be the password assigned by the system. Note for each item there is a associated password.

Finally, the user can carry out operations *insert*, *modify* and *delete* by means of XML files. These files are processed by the *"XML Input"* process. This allows insert, modify or delete an entity or entities encoding data in an XML format.

1.3.2 Manager interface

In order to enable the manager to correctly administrate the Web-site, this interface provides several services such as *view*, *insert*, *modify* or *delete* an entity without previous authentication. Only the manager or administrator can connect to Web-site in private mode through a different URL whose access is based on NCSA http. In addition to these services, the manager interface provides another service that allows the administrator to see all errors occurred during a data exchange between `CoLogNetWS` and other Web-site.

In the next figure, it can be seen the interaction among the different processes accessible through the manager internal interface:



- *1.* Access to manager internal interface.
- *2.* View areas.
- *3.* View groups.
- *4.* View people
- *5.* View software.
- *6.* View projects.
- *7.* View events.
- *8.* Insert *"Entity"* / Modify *"Entity"* / Delete *"Entity"* / View related *"Entity"*.
- *9.* Response to insert *"Entity"* / Response to modify *"Entity"* / Response to delete *"Entity"* / Response to view related *"Entity"*.
- *10.* Ask password / Confirm Password / Insert Form / Modify Form / Insert / Modify.
- *11.* XML file input.
- *12.* XML file processing.
- *13.* View errors produced during data sending/reception.

"Entity" can be: *area*, *group*, *person*, *project*, *software* or *event*.

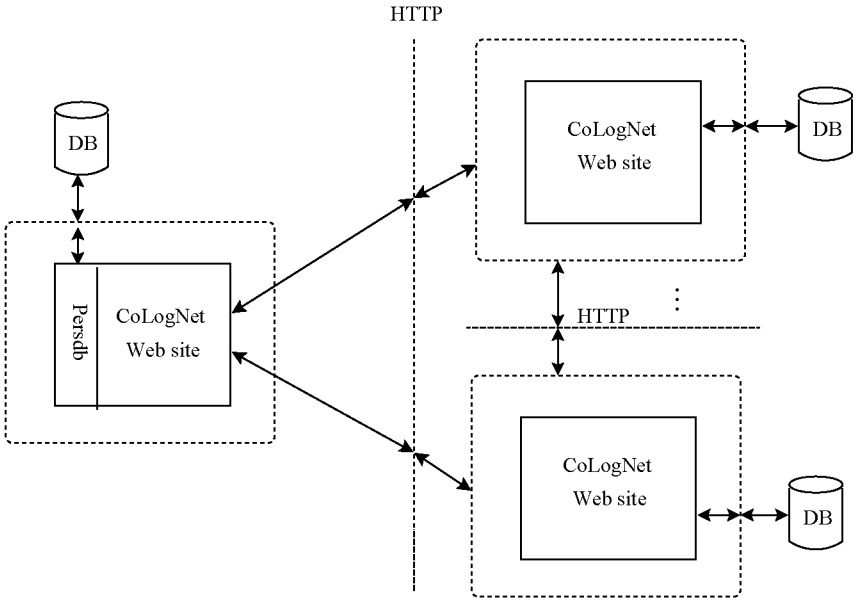
As an user, an administrator accesses to system across WWW interface but in this case, is executed the *"Private Access to Entity"* process. This process allows the manager to have a complete access to the system in a manner similar to the user interface, but on the contrary to the user interface, the manager need not authenticate for executing the operations *modify* and *delete*. An extra operation is that the administrator can check out the status of all carried out transactions between `CoLogNetWS` and other Web-sites, by means of the *"Remote transaction status"* process.

1.4 External interface

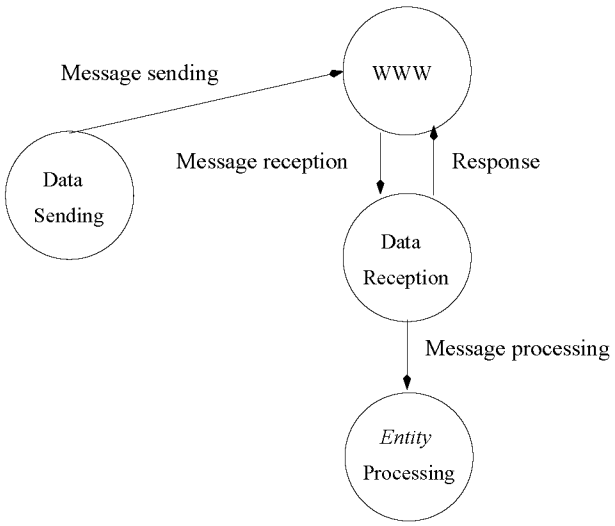
This interface, allows communication between `CoLogNetWS` and other Web-sites. This protocol is based on the HTTP protocol, and the data exchange is encoded in XML format.

Only a new type of message is defined. This message is asynchronous and exchanged by `CoLogNetWS` and the rest of Web-sites in order to update their databases. This message is encoded in XML format and allows *insert*, *modify* and *delete* data.

In this figure is shown the interaction model between `CoLogNetWS` and the rest of Web-sites. As can be seen, `CoLogNetWS` provides its services by encapsulating an XML message into an HTTP request.



In the next figure, we can appreciate the interaction model among the different processes belonging to the external interface:



PART I - Application Core

This part defines the application core, which allows carrying out these operations:

- View the contents of the repository.
- Insert repository data.
- Modify repository data.
- Delete repository data.

The system handles the following entities:

- Areas
- Events
- Groups
- People
- Projects
- Software

2 Public access to entity interface

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#1 (2003/12/5, 12:44:15 CET)

NOTE: The documentation of this application is an abstraction that generalizes all entities stored in the system. The different entities are: groups, areas, events, projects, people, and software.

This application outputs an HTML page showing the items stored in the system for a given entity. It uses the `entity(Entity,Id)` predicate defined in Chapter 4 [Output from entity interface], page 19, which shows all items corresponding to `Entity`, where `Id` is a session identifier created for each execution of this application. In this case, the execution scope associated to the session identifier is `'public'`, so authentication will be required in the operations of modification and deletion.

2.1 Usage and interface (`public_entity`)

- **Library usage:**
  This module is typically compiled as a CGI executable.
- **Other modules used:**
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

2.2 Documentation on internals (`public_entity`)

main/0: PREDICATE

Usage:

- *Description:* Entry predicate to CGI executable.

2.3 Version/Change Log (`public_entity`)

Version 0.1#1 (2003/12/5, 12:44:15 CET)

Started automatic documentation (Jorge Navas)

3 Private access to entity interface

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#2 (2003/12/5, 12:44:43 CET)

NOTE: The documentation of this application is an abstraction that generalizes all entities stored in the system. The different entities are: groups, areas, events, projects, people, and software.

This application outputs an HTML page showing the items stored in the system for a given entity. It uses the `entity(Entity,Id)` predicate defined in Chapter 4 [Output from entity interface], page 19, which shows all items corresponding to `Entity`, where `Id` is a session identifier created for each execution of this application. In this case, the execution scope associated to the session identifier is `'private'`, so authentication will be not required in the operations of modification and deletion.

3.1 Usage and interface (`private_entity`)

- **Library usage:**
  This module is typically compiled as a CGI executable.
- **Other modules used:**
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

3.2 Documentation on internals (`private_entity`)

main/0: PREDICATE

Usage:

- *Description:* Entry predicate to CGI executable.

3.3 Version/Change Log (`private_entity`)

Version 0.1#2 (2003/12/5, 12:44:43 CET)

Started automatic documentation (Jorge Navas)

4 Output from entity interface

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#3 (2003/12/5, 12:45:2 CET)

NOTE: This module is an abstraction that generalizes all entities stored in the system. The different entities are: groups, areas, events, projects, people, and software.

This module provides a predicate to show all items of an entity stored in the system by generating the corresponding HTML code.

4.1 Usage and interface (`entity`)

- **Library usage:**
  `:- use_module(library(entity)).`
- **Exports:**
  - *Predicates:*
    `entity/2`.
- **Other modules used:**
  - *Application modules:*
    `./process_entity.pl`, `../../lib/security/authentication`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

4.2 Documentation on exports (`entity`)

entity/2: PREDICATE

Usage: `entity(+Entity, +Id)`

- *Description:* `Id` is a session identifier associated to current execution. The predicate carries out a query over the database in order to obtain all items stored in the system corresponding to `Entity` and the information associated to each one, and it outputs an HTML page with the content of those information by interpreting the corresponding HTML template.
- *Call and exit should be compatible with:*
  `+Entity` is an entity. (`entity/1`)
  `+Id` is a session identifier. (`id_session/1`)

4.3 Version/Change Log (`entity`)

Version 0.1#3 (2003/12/5, 12:45:2 CET)

Started automatic documentation (Jorge Navas)

5 Entity processing

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#4 (2003/12/5, 12:45:26 CET)

NOTE: This module is an abstraction that generalizes all entities stored in the system. The different entities are: groups, areas, events, projects, people, and software.

Module that defines operations related with the handling of all entities. These operations are: insertion, modification and deletion. Besides, these operations imply authentication process, input operations from HTML forms and output operations to XML files and HTML code through the CGI protocol.

5.1 Usage and interface (process_entity)

- **Library usage:**
  This module can be compiled as a CGI executable or used as a common library.
- **Exports:**
  - *Predicates:*
    `main/1`, `process_entity/4`, `process_entity/5`.
  - *Regular Types:*
    `op_type/1`, `entity/1`, `response_op/1`.
- **Other modules used:**
  - *Application modules:*
    `../../lib/pillow_ext/pillow_ext`, `../structs.pl`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

5.2 Documentation on exports (process_entity)

main/1: PREDICATE

Usage:

- *Description:* Allows to process an item corresponding to an entity through the `process_entity/4` predicate, where the input data is previously provided and the output data is shown by the use of predicates in the `PiLLoW` library.

process_entity/4: PREDICATE

This predicate allows to insert, modify or delete an item corresponding to an entity as well as carrying out the implied operations. These operations are: authentication processes, input operations and output operations.

Usage 1: `process_entity(+Entity, +Op, +Input, ?(Output))`

– *Description:* When `Op` is `'ask_password'`, an authentication process (request of password) is started to access the operations of modification and deletion. This process is only carried out if the execution scope is `'public'`. `Output` is an HTML template to ask for the password of the item corresponding to `Entity` and given by `Input`.

– *Call and exit should be compatible with:*

`+Entity` is an entity. (`entity/1`)

`+Op` and `ask_password` unify. (`= /2`)

`+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

`?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

Usage 2: `process_entity(+Entity, +Op, +Input, ?(Output))`

– *Description:* When `Op` is `'confirm_password'`, the authentication process (confirmation of password) is completed. Checks out that the introduced password in the previous phase matches with the inserted password in the item registration corresponding to `Entity` and given by `Input`. If the authentication is correct, it will pass to the next phase that is indicated by one of the arguments of `Input` that will be modification or deletion of the item. If the authentication is not correct, an HTML template returned in `Output` will repeat the authentication process.

– *Call and exit should be compatible with:*

`+Entity` is an entity. (`entity/1`)

`+Op` and `confirm_password` unify. (`= /2`)

`+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

`?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

Usage 3: `process_entity(+Entity, +Op, +Input, ?(Output))`

– *Description:* When `Op` is `'show'`, all items related with the current item in `Input` (corresponding to `Entity`) will be shown by the HTML template returned in `Output`.

– *Call and exit should be compatible with:*

`+Entity` is an entity. (`entity/1`)

`+Op` and `show` unify. (`= /2`)

`+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

`?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

Usage 4: `process_entity(+Entity, +Op, +Input, ?(Output))`

– *Description:* When `Op` is `'delete'`, the item given by `Input` and corresponding to `Entity` is deleted from the system. Firstly, if the execution scope is `'public'`, it is checked out that the authentication process has been properly carried out. Then, the item is deleted from the system by the `process_entity/5` predicate defined in

this module. `Output` is an HTML template to show the result of the operation. Additionally an XML file is generated by the `code_entity_xml/5` predicate defined in Chapter 13 [XML code generation], page 73. This file contains the data of the item deletion. Depending on the system configuration, this file will be immediately sent or stored in an intermediate buffer.

– *Call and exit should be compatible with:*

`+Entity` is an entity. (`entity/1`)

`+Op` and `delete` unify. (`= /2`)

`+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

`?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

Usage 5: `process_entity(+Entity, +Op, +Input, ?(Output))`

– *Description:* When `Op` is `'register'`, the HTML template returned in `Output` will show an HTML form, partially filled by `Input`, to allow the user to insert a new item correponding to `Entity`.

– *Call and exit should be compatible with:*

`+Entity` is an entity. (`entity/1`)

`+Op` and `register` unify. (`= /2`)

`+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

`?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

Usage 6: `process_entity(+Entity, +Op, +Input, ?(Output))`

– *Description:* When `Op` is `'modify'`, the HTML template returned in `Output` will show an HTML form filled with the data of the item given by `Input` and corresponding to `Entity` in order to the user can modify that form. If the execution scope is `'public'` it is checked out that the authentication process has been properly carried out.

– *Call and exit should be compatible with:*

`+Entity` is an entity. (`entity/1`)

`+Op` and `modify` unify. (`= /2`)

`+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

`?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

Usage 7: `process_entity(+Entity, +Op, +Input, ?(Output))`

– *Description:* When `Op` is `'response_register'`, the item given by `Input` and corresponding to `Entity` is inserted in the system. The item is inserted in the system by the `process_entity/5` predicate defined in this module. `Output` is an HTML template to show the result of the operation. Additionally an XML file is generated by the `code_entity_xml/5` predicate defined in Chapter 13 [XML code generation], page 73. This file contains the data of the item insertion. Depending on the system configuration, this file will be immediately sent or stored in an intermediate buffer.

– *Call and exit should be compatible with:*

`+Entity` is an entity. (`entity/1`)

`+Op` and `response_register` unify. (`= /2`)

`+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

`?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

Usage 8: `process_entity(+Entity, +Op, +Input, ?(Output))`

– *Description:* When `Op` is `'response_modify'`, the item given by `Input` and corresponding to `Entity` is modified from the system. Firstly, if the execution scope is `'public'`, it is checked out that the authentication process has been properly carried out. Then, the item is modified from the system by the `process_entity/5` predicate defined in this module. `Output` is an HTML template to show the result of the operation. Additionally an XML file is generated by the `code_entity_xml/5` predicate defined in Chapter 13 [XML code generation], page 73. This file contains the data of the item modification. Depending on the system configuration, this file will be immediately sent or stored in an intermediate buffer.

– *Call and exit should be compatible with:*

`+Entity` is an entity. (`entity/1`)

`+Op` and `response_modify` unify. (`= /2`)

`+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

`?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

process_entity/5: PREDICATE

This predicate allows to insert, modify or delete an item belonging to an entity without carrying out another auxiliary operations.

Usage 1: `process_entity(+Entity, +Op, +Input, ?(Output), -Res)`

– *Description:* When `Op` is `'delete'`, the item given by `Input` and corresponding to `Entity` is deleted from the system. If `Res` is `'ok'` the operation has been properly carried out, and an HTML template returned in `Output` will show the final information of the deleted item. On the contrary, if `Res` is `'error'` the item has not been deleted because of the existence of another items related with it.

– *Call and exit should be compatible with:*

`+Entity` is an entity. (`entity/1`)

`+Op` and `delete` unify. (`= /2`)

`+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

`?(Output)` is the union of the types: `output_area_struct/1`, `output_event_struct/1`, `output_group_struct/1`, `output_person_struct/1`, `output_project_struct/1` and `output_software_struct/1`. (`output_entity_struct/1`)

`-Res` indicates whether the result of the operation is correct or not. (`response_op/1`)

Usage 2: `process_entity(+Entity, +Op, +Input, ?(Output), ?(Res))`

- *Description:* When `Op` is `'response_register'`, the item given by `Input` and corresponding to `Entity` is inserted in the system. An HTML template returned in `Output` will show the final information of the inserted item. `Res` is not used.
- *Call and exit should be compatible with:*

  `+Entity` is an entity. (`entity/1`)

  `+Op` and `response_register` unify. (`= /2`)

  `+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

  `?(Output)` is the union of the types: `output_area_struct/1`, `output_event_struct/1`, `output_group_struct/1`, `output_person_struct/1`, `output_project_struct/1` and `output_software_struct/1`. (`output_entity_struct/1`)

  `?(Res)` indicates whether the result of the operation is correct or not. (`response_op/1`)

Usage 3: `process_entity(+Entity, +Op, +Input, ?(Output), ?(Res))`

- *Description:* When `Op` is `'response_modify'`, the item given by `Input` and corresponding to `Entity` is modified from the system. An HTML template returned in `Output` will show the final information of the modified item. `Res` is not used.
- *Call and exit should be compatible with:*

  `+Entity` is an entity. (`entity/1`)

  `+Op` and `response_modify` unify. (`= /2`)

  `+Input` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

  `?(Output)` is the union of the types: `output_area_struct/1`, `output_event_struct/1`, `output_group_struct/1`, `output_person_struct/1`, `output_project_struct/1` and `output_software_struct/1`. (`output_entity_struct/1`)

  `?(Res)` indicates whether the result of the operation is correct or not. (`response_op/1`)

op_type/1: REGTYPE

Usage: `op_type(Op)`

- *Description:* `Op` is the type of operation.

entity/1: REGTYPE

Usage: `entity(E)`

- *Description:* `E` is an entity.

response_op/1: REGTYPE

Usage: `response_op(R)`

- *Description:* `R` indicates whether the result of the operation is correct or not.

5.3 Version/Change Log (process_entity)

Version 0.1#4 (2003/12/5, 12:45:26 CET)
Started automatic documentation (Jorge Navas)

PART II - I/O Interface

This part includes a set of modules that define both the internal interface and external interface.

The internal interface is a Web interface which is based on aspects such as interpretation and generation of HTML code, and another related aspects as security. Additionally, an XML interface is offered.

The external interface is based on modules which allow exchanging data with other Web sites. This data exchange is carried out by HTTP protocol and data is encoded in XML format.

6 HTML code generation

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#5 (2003/12/5, 12:46:13 CET)

This library defines predicates for the HTML code generation. Basically it generates lists of items and HTML form elements.

6.1 Usage and interface (code_html)

- **Library usage:**
  `:- use_module(library(code_html)).`
- **Exports:**
  - *Predicates:*
    `keys_HTMLItems/3`, `to_select_default/3`, `predicate_listref/2`, `predicate_ref/3`.
  - *Multifiles:*
    `$is_persistent/2`.
- **Other modules used:**
  - *Application modules:*
    `../../settings`, `../../database/database`, `../../src/entity/process_entity`, `../pillow_ext/pillow_ext`.
  - *System library modules:*
    `pillow/http`, `pillow/html`, `persdb/persdbrt`, `terms`, `file_utils`, `write`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

6.2 Documentation on exports (code_html)

keys_HTMLItems/3: PREDICATE

Usage: `keys_HTMLItems(+Entity, +Keys, ?(HTMLItems))`

- *Description:* Formats a list of identifiers (`Keys`) to a list of HTML items (`HTMLItems`) for the `Entity` entity.
- *Call and exit should be compatible with:*

`+Entity` is an entity.	(`entity/1`)
`+Keys` is a list of `id_htmls`.	(`list/2`)
`?(HTMLItems)` is a list of `item_htmls`.	(`list/2`)

to_select_default/3: PREDICATE

Usage: `to_select_default(+Selected_Items, +Items, ?(HTMLRadio))`

– *Description:* Formats a list of items (`Items`) to HTML form elements of *radio* type (`HTMLRadio`). `Selected_Items` indicates which elements of that *radio* type are selected.
– *Call and exit should be compatible with:*

`+Selected_Items` is a list of `terms`. (`list/2`)

`+Items` is a list of `terms`. (`list/2`)

`?(HTMLRadio)` is a list of `radio_htmls`. (`list/2`)

predicate_listref/2: PREDICATE

Usage: `predicate_listref(+Items, ?(HTMLItemsRef))`

– *Description:* Formats a list of items (`Items`) to a list of HTML items (`HTMLItemsRef`) in which each item is an hyperlink.
– *Call and exit should be compatible with:*

`+Items` is a list of `ref_htmls`. (`list/2`)

`?(HTMLItemsRef)` is a list of `item_htmls`. (`list/2`)

predicate_ref/3: PREDICATE

Usage: `predicate_ref(+Items, ?(Nl), ?(HTMLRefs))`

– *Description:* Formats a list of items (`Items`) to a list of HTML hyperlinks separated by `Nl` delimitator.
– *Call and exit should be compatible with:*

`+Items` is a list of `ref_htmls`. (`list/2`)

`?(Nl)` is an atom. (`atm/1`)

`?(HTMLRefs)` is a list of `terms`. (`list/2`)

6.3 Documentation on multifiles (`code_html`)

$is_persistent/2: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

6.4 Documentation on internals (`code_html`)

id_html/1: REGTYPE

Usage: `id_html(Id)`

– *Description:* `Id` is an identifier associated to an entity.

item_html/1: REGTYPE

Usage: `item_html(Item)`

– *Description:* `Item` is a `PiLLoW` term representing an HTML element of the *'item'* type.

radio_html/1: REGTYPE

Usage: `radio_html(R)`

– *Description:* `R` is an atom representing an HTML element of the *'radio'* type.

ref_html/1: REGTYPE

Usage: `ref_html(R)`

– *Description:* `R` is a term that contains the entity name and its associated URL.

6.5 Version/Change Log (`code_html`)

Version 0.1#5 (2003/12/5, 12:46:13 CET)

Started automatic documentation (Jorge Navas)

7 Security

Version: 0.1#27 (2003/12/5, 13:14:2 CET)
Version of last change: 0.1#6 (2003/12/5, 12:46:48 CET)
This library implements a set of predicates to implement security in the system.

7.1 Security requirements

The security requirements are given, on the one hand, by the functionalities that the application offers and on the other hand, by its own Web nature. About the functionalities, there are public and private accesses, and each one have different needs. About the application nature, this has an interface accessible by WWW, so it is necessary to take into account some aspects to give it the adequate security level.

7.2 CoLogNet Web security structure

To access to whatever of the CoLogNet nodes exists a previous authentication process based on NCSA http. This authentication process is simple, based on the conformity to a certain password. Once the user access to CoLogNet, he or she is in disposition of accessing to the application. This application offers two different scopes or views with their respective security levels, that can be:

- *Public scope* in which any user can access if he or she has previously accessed to CoLogNet, and requires certain control in some operations. Specifically, both deleting and modifying an entity implies a process of password request/confirmation.
- *Private scope* in which an authentication process based on NCSA http must be satisfactorily passed and later no control is required on operation. Specifically any entity can be deleted and modified with no restriction any.

7.3 Security methods

The technique used for authentication is the password request/confirmation. Each CoLogNet user has a password that allows him to access the "net", and also a password associated to each item corresponding to an entity. This password can be randomly generated by the application or given by the own user, and allows the user to delete or modify an item corresponding to an entity. Other security techniques like integrity, confidentiality and non repudation are not implemented in this application.

Besides authentication, another important security aspect implemented in this module, is how the application can decide in each moment in which execution scope is in, and ensure that there is not a fraudulent use of the execution scopes. A typical fraudulent use would be the manipulation of the URL, changing its arguments to avoid the authentication process. For it, it is necessary to implement a mechanism that does not allow these fraudulent uses. This security mechanism is implemented by associating each execution to a session identifier. Each session identifier is related to an execution scope (`'public'` or `'private'`) and an authentication that tells whether that session has been authenticated or not. This mechanism avoids both that an user skips the authentication process and that it accesses to the web through the private view without having the adequate permission.

7.4 Usage and interface (authentication)

- **Library usage:**
  `:- use_module(library(authentication)).`
- **Exports:**
  - *Predicates:*
    `insert_scope/2`, `check_scope/2`, `delete_scope/1`, `init_scope/0`, `insert_authentication/2`, `check_authentication/1`, `delete_authentication/1`, `init_authentication/0`.
  - *Regular Types:*
    `id_session/1`, `scope/1`, `response_authentication/1`.
  - *Multifiles:*
    `$is_persistent/2`.
- **Other modules used:**
  - *System library modules:*
    `persdb/persdbrt`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

7.5 Documentation on exports (authentication)

insert_scope/2: PREDICATE

Usage: `insert_scope(+Id, +S)`

- *Description:* Associates to session identifier `Id`, the execution scope `S`.
- *Call and exit should be compatible with:*
  `+Id` is a session identifier. (`id_session/1`)
  `+S` is the execution scope, which can be `'public'` or `'private'`. (`scope/1`)

check_scope/2: PREDICATE

Usage: `check_scope(?(Id), ?(S))`

- *Description:* Retrieves the execution scope `S` associated to the session identifier `Id`.
- *Call and exit should be compatible with:*
  `?(Id)` is a session identifier. (`id_session/1`)
  `?(S)` is the execution scope, which can be `'public'` or `'private'`. (`scope/1`)

delete_scope/1: PREDICATE

Usage: `delete_scope(+Id)`

- *Description:* Deletes the session identifier `Id` and its execution scope associated.
- *Call and exit should be compatible with:*
  `+Id` is a session identifier. (`id_session/1`)

init_scope/0: PREDICATE

Usage:

- *Description:* Deletes all session identifiers and their associated execution scopes from the system.

insert_authentication/2: PREDICATE

Usage: `insert_authentication(+Id, +A)`

- *Description:* Associates to session identifier `Id` whether has been authenticated or not (`A`).
- *Call and exit should be compatible with:*

  `+Id` is a session identifier. (`id_session/1`)

  `+A` states whether the authentication has been carried out or not. (`response_authentication/1`)

check_authentication/1: PREDICATE

Usage: `check_authentication(+Id)`

- *Description:* Checks out that session identifier `Id` has been properly authenticated.
- *Call and exit should be compatible with:*

  `+Id` is a session identifier. (`id_session/1`)

delete_authentication/1: PREDICATE

Usage: `delete_authentication(+Id)`

- *Description:* Deletes the association between a session identifier `Id` and its authentication.
- *Call and exit should be compatible with:*

  `+Id` is a session identifier. (`id_session/1`)

init_authentication/0: PREDICATE

Usage:

- *Description:* Deletes all associations among session identifiers and their authentications.

id_session/1: REGTYPE

Usage: `id_session(Id)`

- *Description:* `Id` is a session identifier.

scope/1: REGTYPE

Usage: `scope(S)`

- *Description:* `S` is the execution scope, which can be `'public'` or `'private'`.

response_authentication/1: REGTYPE

Usage: `response_authentication(A)`

– *Description:* `A` states whether the authentication has been carried out or not.

7.6 Documentation on multifiles (`authentication`)

$is_persistent/2: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

7.7 Documentation on internals (`authentication`)

scope_def/2: PREDICATE

`scope_def(Id, S)`

`Id` is a session identifier and `S` is the execution scope associated to that identifier.

The predicate is of type *data*.

authentication/2: PREDICATE

`authentication(Id, A)`

`Id` is a session identifier and `A` indicates whether that session has been authenticated or not.

The predicate is of type *data*.

7.8 Version/Change Log (`authentication`)

Version 0.1#6 (2003/12/5, 12:46:48 CET)

Started automatic documentation (Jorge Navas)

8 Information exchange protocol

8.1 Introduction

This chapter defines an Information Exchange Protocol among the different Web sites that belong to the CoLogNet Network. It is intended that all Web sites in the Colognet Network use this protocol to exchange data about groups, researchers, etc.

8.2 CoLogNet Web structure

The Colognet network consists of a set of specific Web sites related to the different areas in which the network is split. Each Web site may contain a user-updatable database to store data regarding groups, people, projects, etc, working in each area. The Information Exchange Protocol here defined will make possible the automatic exchange of data between all Colognet Web sites, in order to keep databases up-to-date among the different Web sites.

8.3 Protocol Definition

The communication protocol between the different Web sites will be based on the HTTP protocol, and the information exchanged will be encoded in the XML format.

8.3.1 Information sending

The information will be sent from a Web site to others as if it were the data coming from an HTML form whose handler is located in the destination Web site. The mimicked form would have an input field of type "file" and name "filen", and forced by such input field type the encoding of the data ("ENCTYPE") will be "multipart/form-data".

The process will require the establishment of a TCP connection to port number 80 of the destination Web site (the HTTP server port). After establishing the connection with the HTTP server, the sender application will send data composed by the required HTTP headers and a body which will contain an XML file, where the exchanged data will be encoded. The XML file should be in accordance with the XML Scheme Definition (XSD) defined in Chapter 11 [XSD associated to the XML data], page 49. After receiving and processing the sent data, if everything is correct the HTTP server should return an OK response with code 200 and the value of the `'colognet_error'` header would be `'not_error'`. Another response code or header value will indicate an error, to be exceptionally handled.

This new HTTP header is defined due to the fact that is not enough a HTTP server response with code 200 for knowing if the data has been correctly processed. The problem is that two sorts of errors can be produced in an information exchange: HTTP server error and XML file processing error. The first error is defined with code 500 in the HTTP protocol. However the second error is not defined in this protocol, so we need another way of signaling an error on XML file.

A solution would be to define a new HTTP code, but this it is not possible in this protocol. Due to this, we have decided to define a new HTTP header, which will be added to HTTP response and will point out the result of processing an XML file. This HTTP header will be named `'colognet_error'`. The current values of `'colognet_error'` are:

- *error_entity_integrity* states that there are repeated keys in the XML file.
- *error_reference_integrity* states that is not accomplished the reference integrity in the XML file.
- *error_empty* states that there are empty required fields in the XML file.

- *object_not_found* states that an entity has not been found.
- *error_invalid_syntax* states that the XML file is not correctly formed.
- *error_category_event* states that the event category is not valid.
- *error_category_group* states that the group category is not valid.
- *error_date_incorrect* states that there is an invalid date.
- *error_not_relations* states that an item can not be deleted due to there are other items belonging to it.
- *not_error* states that the XML data processing has been correct.

Note that you can define new values of the HTTP header "colognet_error", if you need them.

8.3.2 Example of HTTP request

In the following an example is shown in order to make clear the data which is sent, using the HTTP protocol, in a database update. The data corresponds to a registration of a research group. Note that the `Content-Length` header has to match with the length in bytes of the data following the first blank line. Also note that the boundary can be changed, and should not occur inside the XML data.

```
POST /~clip/Projects/COLOGNET/cgi-bin/exchange/recv.cgi HTTP/1.0
Content-Type: multipart/form-data; boundary="6G+f"
Content-Length: 603

--6G+f

Content-Disposition: form-data; name="filen"
Content-Type: text/xml

<register>
 <object>
  <group>
     <insta>UPM</insta>
     <instf>Universidad Politecnica de Madrid</instf>
     <wwwi>http://www.upm.es</wwwi>
     <groupa>CLIP</groupa>
     <groupf>Computing Logic: Implementation and Paralelism</groupf>
     <wwwg>http://clip.dia.fi.upm.es</wwwg>
     <category>University </category>
     <description> </description>
     <address>Campus MonteGancedo </address>
     <zip>28660</zip>
     <city>Boadilla del Monte, Madrid</city>
     <state> </state>
     <country>Spain</country>
     <password>clip </password>
     <member></member>
     <contact_person></contact_person>
     <coordinating></coordinating>
     <related></related>
     <working></working>
  </group>
```

```
 </object>
 <associated_data></associated_data>
</register>

--6G+f--
```

8.3.3 Information reception

The reception process is equivalent to the reception by a CGI form handler of the data coming from a form, with the characteristics mentioned above.

8.3.4 Example of HTTP response

In the following an example is shown in order to make clear the data which is sent back, using the HTTP protocol, in a succesful database update. Note the use of the HTTP header `'colognet_error'`:

```
HTTP/1.1 200 OK
Date: Fri, 24 Oct 2003 17:33:01 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Colognet_Error: not_error
Content-Length: 0
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

9 Reception of data

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#26 (2003/12/5, 13:13:39 CET)

This application, which behaves as a CGI executable, implements the reception of data, encoded in XML format. This data can come from the own Web site or from other Web sites, thus allowing the communication among the databases of the different nodes corresponding to CoLogNet. Both ways of communication are based on the HTTP protocol.

9.1 Usage and interface (`recv`)

- **Library usage:**
  This module is typically compiled as a CGI executable.
- **Other modules used:**
  - *Application modules:*
    `../../src/areas/process_areas`, `../../src/events/process_events`, `../../src/groups/process_groups`, `../../src/people/process_people`, `../../src/projects/process_projects`, `../../src/software/process_software`, `../../src/structs`, `../../settings`, `../pillow_ext/pillow_ext`, `../xml/error_xml`, `./send`.
  - *System library modules:*
    `pillow/http`, `pillow/html`, `file_utils`, `lists`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

9.2 Documentation on internals (`recv`)

main/1: PREDICATE

Usage:

- *Description:* Main entry of the application. Allows to carry out the data reception encoded in XML format, through the HTTP protocol. Once the XML file is received, it is processed and the corresponding response is sent to the sender Web site.

ask_file/1: PREDICATE

Usage: `ask_file(?(Output))`

- *Description:* `Output` refers to the Web page created to ask the user for the XML file which will be processed.
- *Call and exit should be compatible with:*
  `?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

process_file/3: PREDICATE

Once the content of the XML file is received, this file is parsed and validated, and the values of its different elements are retrieved, in order to this data be processed.

To better explain the different steps, we show a possible XML file, where the more important parts are indicated:

```
<register>        <!-- Level 1-->
<object>          <!-- Level 1.1-->
 <group>          <!-- Level 1.1.1-->
   <identifier>57</identifier>
   <insta>UPM</insta>
   <instf>Universidad Politecnica de Madrid</instf>
   <wwwi>http://www.upm.es</wwwi>
   <groupa>CLIP</groupa>
   <groupf>CLIP Lab</groupf>
   <wwwg>http:// clip.dia.fi.upm.es</wwwg>
   <category>University</category>
   <description></description>
   <address>Campus MonteGancedo</address>
   <zip>28660</zip>
   <city>Boadilla del Monte, Madrid</city>
   <state></state>
   <country>Spain</country>
   <password>CLIP</password>
   <member><person>56</person></member>
   <contact_person><person>56</person></contact_person>
   <coordinating><project>62</project></coordinating>
   <related><software>0</software></related>
   <working>
    <area>0</area>
   </working>
 </group>
</object>
<associated_data> <!-- Level 1.2-->
  <person>        <!-- Level 1.2.1.-->
    <identifier>56</identifier>
    <last_name>Cabeza</last_name>
    <first_name>Daniel</first_name>
    <password>bardo</password>
  </person>
  <project>       <!-- Level 1.2.2 -->
    <identifier>62</identifier>
    <titlea>CoLogNet</titlea>
    <titlef>CoLogNet</titlef>
    <password>colognet</password>
  </project>
  <software>      <!-- Level 1.2.3 -->
    <identifier>0</identifier>
    <namea>Ciao</namea>
    <namef>The Ciao Program Development System</namef>
    <password>ciao</password>
```

```
  </software>
  <area>          <!-- Level 1.2.4 -->
    <identifier>0</identifier>
    <namea>Automatic Deduction Systems</namea>
    <namef>Automated Deduction Systems and Theorem Provers</namef>
    <password>jorge</password>
  </area>
</associated_data>
</register>
```

The relevant points are:

- **Level 1** element. Allows to know if the operation is an insertion (register), modification (modify) or deletion (unregister).
- **Level 1.1** and **level 1.2** elements. The **Level 1.1** is defined by the "object" node and contains all entities on which the operation is carried out. Those entities compose the **level 1.1.x**. On the other hand, the **level 1.2** is defined by the "associated_data" node and contains all auxiliary entities to the **level 1.1.x** entities. Those auxiliary entities make up the **level 1.2.x**.

The Chapter 11 [XSD associated to the XML data], page 49 explains the composition of all the possible XML files.

To process the XML file, two stages clearly differenced are carried out. On the one hand, the information associated to the **level 1.2.x** is parsed, validated by a set of checks, and finally is retrieved. The checks carried out are:

- Unique identifiers. Except for the event entity, since this entity has not defined any identifier.
- Existence of the auxiliary entities.
- The required fields must be filled.

On the other hand, the data associated to the **level 1.1.x** is also parsed, validated and retrieved. This second step is only carried out if the previous checks have not detected any error. It is very important, in this case, establishing a correct order in the validation of entities, to avoid inconsistent situations. That is, first the less related entities are validated, and later the more related ones. The correct order would be: events, areas, projects, software, people and groups. The checks of the **level 1.1.x** are:

- Existence of the entity. In case of a modification or deletion.
- The required fields must be filled.
- The referential integrity must be accomplished. Except in the case of deletion.
- The category must be correct. In case it is a group or event.
- The dates must be correct. In case it is an event.

Finally if no error has been produced, the validated information is processed, that is, the data is inserted, modified or deleted.

Usage 1: `process_file(+Op, +Input, ?(Output))`

– *Description:* When `Op` is `'register_file'`, this service allows to insert, modify or delete an entity (or entities) by means of an XML file given in `Input`, from the same Web site in which the XML file will be processed. `Output` is an HTML template that will show the result of processing the XML file. Depending on the system configuration, the XML file will be immediately sent to the rest of Web sites or stored in an intermediate buffer.

– *Call and exit should be compatible with:*

`+Op` and `register_file` unify. (`= /2`)

`+Input` is a term that stores the content of a file. (`input_file_struct/1`)

`?(Output)` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template. (`output_template/1`)

Usage 2: `process_file(+Op, +Input, ?(Output))`

– *Description:* When `Op` is `'register_nodes'` this service allows to insert, modify or delete an entity (or entities) by means of an XML file given in `Input`, from other Web site. `Output` is the response of proccessing that XML file.

If there has not been any error, the value of `'colognet_error'` header contained in `Output`, is `'not_error'`. This header is added to the rest of HTTP headers that the server sends back to the sender Web site. If there has been some error, the same process is carried out adding the previous header with the value corresponding to the type of error. Also the cause of the error is added to the HTTP response body. So, it is very important to add the `'colognet_error'` header, because the sender Web site need it to know the result of the operation.

– *Call and exit should be compatible with:*

`+Op` and `register_nodes` unify. (`= /2`)

`+Input` is a term that stores the content of a file. (`input_file_struct/1`)

`?(Output)` is a list of `html_terms`. (`list/2`)

9.3 Version/Change Log (recv)

Version 0.1#26 (2003/12/5, 13:13:39 CET)

Started automatic documentation (Jorge Navas)

10 Sending of data

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

This module implements the sending of data, encoded in XML format to other Web sites by the HTTP protocol, allowing by this way the communication between the databases corresponding to CoLogNet. Depending on the system configuration, the information will be immediately sent to the rest of Web sites or stored in an intermediate buffer.

The first type of sending is used when the data traffic is reduced or an inmediate update is required. On the contrary, the second type of sending is thought to avoid the traffic congestion among the different Web sites. For this reason, it is executed periodically by a daemon. Therefore, the XML files are put in intermediate buffers and are not immediately sent. The daemon will send a burst of data at hours determined by the administrator.

10.1 Usage and interface (send)

- **Library usage:**
  This module is compiled as an executable or can be used as a library.
- **Exports:**
  - *Predicates:*
    `main/0`, `data_exchange/3`.
- **Other modules used:**
  - *Application modules:*
    `../pillow_ext/http_aux`, `../pillow_ext/pillow_ext`, `../util/url_websites`, `../xml/error_xml`, `../../src/entity/process_entity`, `../../settings`.
  - *System library modules:*
    `pillow/http`, `pillow/html`, `system`, `file_utils`, `lists`, `sort`, `filenames`, `pillow/http_ll`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

10.2 Documentation on exports (send)

main/0: PREDICATE

Usage:

- *Description:* Implements the process of sending of data encoded in XML format by the HTTP protocol. To this end, it obtains all the XML files stored in the intermediate buffers using the `directory_files_mapped/2` predicate, and then the files are sent in date order to the rest of the Web sites by the `send_files/1` predicate. Both predicates are defined in this module.

data_exchange/3: PREDICATE

Usage: `data_exchange(?(Buffer), +Message, ?(Entity))`

– *Description:* Allows to send an XML file, inmediately or in batch mode, to the rest of Web sites. The way of choosing one of the two types is defined in a configuration file, called `settings.pl`.

  If the sending of data is inmediate, this predicate sends at once a XML message `Message` to the rest of Web sites. These Web sites are defined by the `url_websites` predicate.

  On the contrary, if the sending of data is in batch mode, it creates a file with the content of `Message` regarding an entity `Entity`, in the `Buffer` directory. Specifically it creates an XML file and puts it in an intermediate buffer.

– *Call and exit should be compatible with:*

`?(Buffer)` is an intermediate buffer.	(`buffer/1`)
`+Message` is a message in XML format.	(`xml_message/1`)
`?(Entity)` is an entity.	(`entity/1`)

10.3 Documentation on internals (send)

directory_files_mapped/2: PREDICATE

Usage: `directory_files_mapped(+Buffer, ?(Files_Mapped))`

– *Description:* Provides in `Files_Mapped` data about all the files stored in the buffer `Buffer`.

– *Call and exit should be compatible with:*

`+Buffer` is an intermediate buffer.	(`buffer/1`)
`?(Files_Mapped)` is a list of `file_mapped`s.	(`list/2`)

send_files/1: PREDICATE

Usage: `send_files(+Files)`

– *Description:* Sends the XML files `Files` from the intermediate buffers to the rest of Web sites with which exists a communication among their databases. Firstly, the Web sites URLs are obtained. These URLs are obtained by the `url_websites/2` predicate defined in the `url_websites` file. This predicate allows to obtain the URL depending on the type of entity that has generated the XML file. Later a HTTP request is created (see Section 8.3.2 [Example of HTTP request], page 38), and is sent to each URL. Finally the server response is analyzed:

  - If a server error is produced (code 500), the file is not removed, waiting for being sent again. This is because of a temporal fault of the receiving server or problems in the connection.
  - If the server response is correct (code 200) the `'cognet_error'` header is analyzed to verify whether the transaction has been done correctly, or the type of error in case that it has failed. If there has not been any error, the transaction is considered as finished and the file is deleted after it has been sent to the last Web site. In other case, the system obtains information about the type of error, which is written in a log file, and the XML file is moved to a special directory to be able later analyzed the cause of the error.

– *Call and exit should be compatible with:*

`+Files` is a list of `file_mapped`s.	(`list/2`)

buffer/1: REGTYPE

`buffer(B)`

`B` is an intermediate buffer managed by the system to store temporally the XML files, generated by the different transactions carried out in the Web site. The different types of buffers are:

- `areas` stores the XML files associated to areas.
- `events` stores the XML files associated to events.
- `groups` stores the XML files associated to groups.
- `people` stores the XML files associated to people.
- `projects` stores the XML files associated to projects.
- `software` stores the XML files associated to software.
- `all` stores the XML files received from the own Web site.

Usage: `buffer(B)`

– *Description:* `B` is an intermediate buffer.

file_mapped/1: REGTYPE

`file_mapped(F)`

`F` contains information about an XML file. This information is a tuple *s(Time,Name,Entity)* where:

- *Time* is the date when the XML file was created.
- *Name* is the complete path of the XML file.
- *Entity* is the XML file entity.

Usage: `file_mapped(F)`

– *Description:* `F` contains information about an XML file.

xml_message/1: REGTYPE

Usage: `xml_message(M)`

– *Description:* `M` is a message in XML format.

10.4 Version/Change Log (send)

Version 0.1#27 (2003/12/5, 13:14:2 CET)

Started automatic documentation (Jorge Navas)

11 XSD associated to the XML data

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#7 (2003/12/5, 12:47:36 CET)

The communication among the different Web sites is carried out by a data exchange that allows to keep updated among themselves their databases. This exchange data is encoded in XML format. Specifically three different XSD (XML Scheme Definition) have been defined for each type of transaction: insertion, modification and deletion.

11.1 Insertion XSD

```
 <?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="TypesScheme"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:register="TypesScheme" elementFormDefault="qualified"
 attributeFormDefault="unqualified">

<!-- Simples Types Declaration -->

<xs:simpleType name="email">
  <xs:restriction base="xs:string">
    <xs:pattern value="(\w|_)+@((\w|_)+\.)+(\w|_)+"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="not_empty">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse"/>
    <xs:pattern value="(.)+"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="category_group">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Company"/>
    <xs:enumeration value="Non profit research institute"/>
    <xs:enumeration value="Research institute"/>
    <xs:enumeration value="University"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="category_event">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Workshop"/>
    <xs:enumeration value="Conference"/>
    <xs:enumeration value="Symposium"/>
    <xs:enumeration value="Congress"/>
  </xs:restriction>
</xs:simpleType>

<!-- Global Elements Definition-->
```

```
<xs:element name="password" type="register:not_empty"/>
<xs:element name="identifier" type="xs:integer"/>

<!-- Global Groups Definition-->
<xs:group name="location">
  <xs:sequence>
    <xs:element name="address" type="register:not_empty"/>
    <xs:element name="zip" type="register:not_empty"/>
    <xs:element name="city" type="register:not_empty"/>
    <xs:element name="state" type="xs:string"/>
    <xs:element name="country" type="register:not_empty"/>
  </xs:sequence>
</xs:group>

<!-- General Definition -->
<xs:element name="register">
<xs:complexType>
<xs:sequence>
<xs:element name="object">
<xs:complexType>
<xs:sequence>

<xs:element name="group" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="insta" type="register:not_empty"/>
<xs:element name="instf" type="register:not_empty"/>
<xs:element name="wwwi" type="register:not_empty"/>
<xs:element name="groupa" type="register:not_empty"/>
<xs:element name="groupf" type="register:not_empty"/>
<xs:element name="wwwg" type="register:not_empty"/>
<xs:element name="category" type="register:category_group"/>
<xs:element name="description" type="xs:string"/>
<xs:group ref="register:location"/>
<xs:element ref="register:password"/>
<xs:element name="member">
<xs:complexType>
<xs:sequence>
<xs:element name="person" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="contact_person">
<xs:complexType>
<xs:sequence>
<xs:element name="person" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
```

```
</xs:complexType>
</xs:element>

<xs:element name="coordinating">
<xs:complexType>
<xs:sequence>
<xs:element name="project" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="related">
<xs:complexType>
<xs:sequence>
<xs:element name="software" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="working">
<xs:complexType>
<xs:sequence>
<xs:element name="area" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="person" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="last_name" type="register:not_empty"/>
<xs:element name="first_name" type="register:not_empty"/>
<xs:element name="title" type="register:not_empty"/>
<xs:element name="www" type="register:not_empty"/>
<xs:element name="email" type="register:email"/>
<xs:element name="phone" type="xs:positiveInteger"/>
<xs:element name="fax" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:group ref="register:location"/>
<xs:element ref="register:password"/>

<xs:element name="member">
<xs:complexType>
<xs:sequence>
```

```
<xs:element name="group" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="contact_person">
<xs:complexType>
<xs:sequence>
<xs:element name="group" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="contact_software">
<xs:complexType>
<xs:sequence>
<xs:element name="software" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="project" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="titlea" type="register:not_empty"/>
<xs:element name="titlef" type="register:not_empty"/>
<xs:element name="www" type="register:not_empty"/>
<xs:element name="category" type="register:not_empty"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="comment" type="xs:string"/>
<xs:element ref="register:password"/>

<xs:element name="coordinating">
<xs:complexType>
<xs:sequence>
<xs:element name="group" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="software" minOccurs="0" maxOccurs="unbounded">
```

```
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="namea" type="register:not_empty"/>
<xs:element name="namef" type="register:not_empty"/>
<xs:element name="www" type="register:not_empty"/>
<xs:element name="ftp" type="register:not_empty"/>
<xs:element name="platforms" type="register:not_empty"/>
<xs:element name="availability" type="register:not_empty"/>
<xs:element name="description" type="xs:string"/>
<xs:element ref="register:password"/>

<xs:element name="contact_software" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:element name="person" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="related">
<xs:complexType>
<xs:sequence>
<xs:element name="group" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="event" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="title" type="register:not_empty"/>
<xs:element name="category" type="register:category_event"/>
<xs:element name="city" type="register:not_empty"/>
<xs:element name="country" type="register:not_empty"/>
<xs:element name="first_date" type="xs:date"/>
<xs:element name="last_date" type="xs:date"/>
<xs:element name="deadline" type="xs:date"/>
<xs:element name="comment" type="xs:string"/>
<xs:element name="person_contact" type="register:not_empty"/>
<xs:element name="email" type="register:email"/>
<xs:element ref="register:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
<xs:element name="area" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="namea" type="register:not_empty"/>
<xs:element name="namef" type="register:not_empty"/>
<xs:element name="description" type="xs:string"/>
<xs:element ref="register:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="associated_data">
<xs:complexType>
<xs:sequence>

<xs:element name="group" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="wwwi" type="register:not_empty"/>
<xs:element name="wwwg" type="register:not_empty"/>
<xs:element ref="register:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="person" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="last_name" type="register:not_empty"/>
<xs:element name="first_name" type="register:not_empty"/>
<xs:element ref="register:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="project" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="titlea" type="register:not_empty"/>
<xs:element name="titlef" type="register:not_empty"/>
<xs:element ref="register:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
<xs:element name="software" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="namea" type="register:not_empty"/>
<xs:element name="namef" type="register:not_empty"/>
<xs:element ref="register:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="area" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="register:identifier"/>
<xs:element name="namea" type="register:not_empty"/>
<xs:element name="namef" type="register:not_empty"/>
<xs:element ref="register:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

  <xs:key name="GroupObjectKey">
    <xs:selector xpath="register:object/register:group"/>
    <xs:field xpath="register:identifier"/>
  </xs:key>
  <xs:key name="PersonObjectKey">
    <xs:selector xpath="register:object/register:person"/>
    <xs:field xpath="register:identifier"/>
  </xs:key>
  <xs:key name="ProjectObjectKey">
    <xs:selector xpath="register:object/register:project"/>
    <xs:field xpath="register:identifier"/>
  </xs:key>
  <xs:key name="SoftwareObjectKey">
    <xs:selector xpath="register:object/register:software"/>
    <xs:field xpath="register:identifier"/>
  </xs:key>
  <xs:key name="AreaObjectKey">
    <xs:selector xpath="register:object/register:area"/>
    <xs:field xpath="register:identifier"/>
  </xs:key>
  <xs:key name="GroupAssociatedKey">
    <xs:selector xpath="register:associated_data/register:group"/>
    <xs:field xpath="register:identifier"/>
```

```
  </xs:key>
  <xs:key name="PersonAssociatedKey">
    <xs:selector xpath="register:associated_data/register:person"/>
    <xs:field xpath="register:identifier"/>
  </xs:key>
  <xs:key name="ProjectAssociatedKey">
    <xs:selector xpath="register:associated_data/register:project"/>
    <xs:field xpath="register:identifier"/>
  </xs:key>
  <xs:key name="SoftwareAssociatedKey">
    <xs:selector xpath="register:associated_data/register:software"/>
    <xs:field xpath="register:identifier"/>
  </xs:key>
  <xs:key name="AreaAssociatedKey">
    <xs:selector xpath="register:associated_data/register:area"/>
    <xs:field xpath="register:identifier"/>
  </xs:key>
   </xs:element>
</xs:schema>
```

11.2 Modification XSD

```
   <?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="TypesScheme" xmlns:modify="TypesScheme"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">

<!-- Simples Types Declaration -->

<xs:simpleType name="email">
  <xs:restriction base="xs:string">
    <xs:pattern value="(\w|_)+@((\w|_)+\.)+(\w|_)+"/>
  </xs:restriction>

</xs:simpleType>
<xs:simpleType name="not_empty">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse"/>
    <xs:pattern value="(.)+"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="category_group">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Company"/>
    <xs:enumeration value="Non profit research institute"/>
    <xs:enumeration value="Research institute"/>
    <xs:enumeration value="University"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="category_event">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Workshop"/>
    <xs:enumeration value="Conference"/>
    <xs:enumeration value="Symposium"/>
    <xs:enumeration value="Congress"/>
  </xs:restriction>
</xs:simpleType>

<!-- Global Elements Definition-->
<xs:element name="password" type="modify:not_empty"/>
<xs:element name="identifier" type="xs:integer"/>

<!-- Global Groups Definition-->

<xs:group name="location">
  <xs:sequence>
    <xs:element name="address" type="modify:not_empty"/>
    <xs:element name="zip" type="modify:not_empty"/>
    <xs:element name="city" type="modify:not_empty"/>
    <xs:element name="state" type="xs:string"/>
    <xs:element name="country" type="modify:not_empty"/>
  </xs:sequence>
</xs:group>

<!-- General Definition -->
<xs:element name="modify">
<xs:complexType>
<xs:sequence>

<xs:element name="object">
<xs:complexType>
<xs:sequence>

<xs:element name="group" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
<xs:element name="insta" type="modify:not_empty"/>
<xs:element name="instf" type="modify:not_empty"/>
<xs:element name="wwwi" type="modify:not_empty"/>
<xs:element name="wwwg" type="modify:not_empty"/>
<xs:element name="category" type="modify:category_group"/>
<xs:element name="description" type="xs:string" />
<xs:group ref="modify:location"/>
<xs:element name="old_groupa" type="modify:not_empty"/>
<xs:element name="new_groupa" type="modify:not_empty"/>
<xs:element name="old_groupf" type="modify:not_empty"/>
<xs:element name="new_groupf" type="modify:not_empty"/>
<xs:element name="old_password" type="modify:not_empty"/>
```

```
<xs:element name="new_password" type="modify:not_empty"/>

<xs:element name="member">
<xs:complexType>
<xs:sequence>
<xs:element name="person" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="contact_person">
<xs:complexType>
<xs:sequence>
<xs:element name="person" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="coordinating">
<xs:complexType>
<xs:sequence>
<xs:element name="project" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="related">
<xs:complexType>
<xs:sequence>
<xs:element name="software" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="working">
<xs:complexType>
<xs:sequence>
<xs:element name="area" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="person" minOccurs="0" maxOccurs="unbounded">
```

```
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
<xs:element name="title" type="modify:not_empty"/>
<xs:element name="www" type="modify:not_empty"/>
<xs:element name="email" type="modify:email"/>
<xs:element name="phone" type="xs:positiveInteger"/>
<xs:element name="fax" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:group ref="modify:location"/>
<xs:element name="old_last_name" type="modify:not_empty"/>
<xs:element name="new_last_name" type="modify:not_empty"/>
<xs:element name="old_first_name" type="modify:not_empty"/>
<xs:element name="new_first_name" type="modify:not_empty"/>
<xs:element name="old_password" type="modify:not_empty"/>
<xs:element name="new_password" type="modify:not_empty"/>

<xs:element name="member">
<xs:complexType>
<xs:sequence>
<xs:element name="group" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="contact_person">
<xs:complexType>
<xs:sequence>
<xs:element name="group" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="contact_software">
<xs:complexType>
<xs:sequence>
<xs:element name="software" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="project" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
```

```
<xs:element name="www" type="modify:not_empty"/>
<xs:element name="category" type="modify:not_empty"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="comment" type="xs:string" />
<xs:element name="old_titlea" type="modify:not_empty"/>
<xs:element name="new_titlea" type="modify:not_empty"/>
<xs:element name="old_titlef" type="modify:not_empty"/>
<xs:element name="new_titlef" type="modify:not_empty"/>
<xs:element name="old_password" type="modify:not_empty"/>
<xs:element name="new_password" type="modify:not_empty"/>

<xs:element name="coordinating">
<xs:complexType>
<xs:sequence>
<xs:element name="group" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="software" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
<xs:element name="www" type="modify:not_empty"/>
<xs:element name="ftp" type="modify:not_empty"/>
<xs:element name="platforms" type="modify:not_empty"/>
<xs:element name="availability" type="modify:not_empty"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="old_namea" type="modify:not_empty"/>
<xs:element name="new_namea" type="modify:not_empty"/>
<xs:element name="old_namef" type="modify:not_empty"/>
<xs:element name="new_namef" type="modify:not_empty"/>
<xs:element name="old_password" type="modify:not_empty"/>
<xs:element name="new_password" type="modify:not_empty"/>

<xs:element name="contact_software" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:element name="person" type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="related">
<xs:complexType>
<xs:sequence>
```

```
<xs:element name="group"  type="xs:integer"
 minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="event" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="category" type="modify:category_event"/>
<xs:element name="city" type="modify:not_empty"/>
<xs:element name="country" type="modify:not_empty"/>
<xs:element name="first_date" type="xs:date"/>
<xs:element name="last_date" type="xs:date"/>
<xs:element name="deadline" type="xs:date"/>
<xs:element name="comment" type="xs:string"/>
<xs:element name="person_contact" type="modify:not_empty"/>
<xs:element name="email" type="modify:email"/>
<xs:element name="old_title" type="modify:not_empty"/>
<xs:element name="new_title" type="modify:not_empty"/>
<xs:element name="old_password" type="modify:not_empty"/>
<xs:element name="new_password" type="modify:not_empty"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="area" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="old_namea" type="modify:not_empty"/>
<xs:element name="new_namea" type="modify:not_empty"/>
<xs:element name="old_namef" type="modify:not_empty"/>
<xs:element name="new_namef" type="modify:not_empty"/>
<xs:element name="old_password" type="modify:not_empty"/>
<xs:element name="new_password" type="modify:not_empty"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="associated_data">
<xs:complexType>
<xs:sequence>
```

```
<xs:element name="group" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
<xs:element name="wwwi" type="modify:not_empty"/>
<xs:element name="wwwg" type="modify:not_empty"/>
<xs:element ref="modify:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="person" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
<xs:element name="last_name" type="modify:not_empty"/>
<xs:element name="first_name" type="modify:not_empty"/>
<xs:element ref="modify:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="project" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
<xs:element name="titlea" type="modify:not_empty"/>
<xs:element name="titlef" type="modify:not_empty"/>
<xs:element ref="modify:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="software" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
<xs:element name="namea" type="modify:not_empty"/>
<xs:element name="namef" type="modify:not_empty"/>
<xs:element ref="modify:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="area" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element ref="modify:identifier"/>
<xs:element name="namea" type="modify:not_empty"/>
<xs:element name="namef" type="modify:not_empty"/>
<xs:element ref="modify:password"/>
```

```
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

    <xs:key name="GroupObjectKey">
        <xs:selector xpath="modify:object/modify:group"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
    <xs:key name="PersonObjectKey">
        <xs:selector xpath="modify:object/modify:person"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
    <xs:key name="ProjectObjectKey">
        <xs:selector xpath="modify:object/modify:project"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
    <xs:key name="SoftwareObjectKey">
        <xs:selector xpath="modify:object/modify:software"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
    <xs:key name="AreaObjectKey">
        <xs:selector xpath="modify:object/modify:area"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
    <xs:key name="GroupAssociatedKey">
        <xs:selector xpath="modify:associated_data/modify:group"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
    <xs:key name="PersonAssociatedKey">
        <xs:selector xpath="modify:associated_data/modify:person"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
    <xs:key name="ProjectAssociatedKey">
        <xs:selector xpath="modify:associated_data/modify:project"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
    <xs:key name="SoftwareAssociatedKey">
        <xs:selector xpath="modify:associated_data/modify:software"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
    <xs:key name="AreaAssociatedKey">
        <xs:selector xpath="modify:associated_data/modify:area"/>
        <xs:field xpath="modify:identifier"/>
    </xs:key>
   </xs:element>
</xs:schema>
```

11.3 Deletion XSD

```
   <?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="TypesScheme"
 xmlns:unregister="TypesScheme"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">

<!-- Simples Types Declaration -->
<xs:simpleType name="not_empty">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse"/>
    <xs:pattern value="(.)+"/>
  </xs:restriction>
</xs:simpleType>

<!-- Global Elements Definition-->
<xs:element name="password" type="unregister:not_empty"/>

<!-- Entities Definition -->
<xs:element name="unregister">
<xs:complexType>
<xs:sequence>

<xs:element name="object">
<xs:complexType>
<xs:sequence>

<xs:element name="group" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="groupa" type="unregister:not_empty"/>
<xs:element name="groupf" type="unregister:not_empty"/>
<xs:element ref="unregister:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="person" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="last_name" type="unregister:not_empty"/>
<xs:element name="first_name" type="unregister:not_empty"/>
<xs:element ref="unregister:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
<xs:element name="project" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="titlea" type="unregister:not_empty"/>
<xs:element name="titlef" type="unregister:not_empty"/>
<xs:element ref="unregister:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="software" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="namea" type="unregister:not_empty"/>
<xs:element name="namef" type="unregister:not_empty"/>
<xs:element ref="unregister:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="event" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="title" type="unregister:not_empty"/>
<xs:element ref="unregister:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="area" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="namea" type="unregister:not_empty"/>
<xs:element name="namef" type="unregister:not_empty"/>
<xs:element ref="unregister:password"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="associated_data"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

In the following, XML examples are defined for each one of the possible transactions.

11.4 Example of Insertion

```
  <register>
<object>
 <group>
   <identifier>57</identifier>
   <insta>UPM</insta>
   <instf>Universidad Politecnica de Madrid</instf>
   <wwwi>http://www.upm.es</wwwi>
   <groupa>CLIP</groupa>
   <groupf>CLIP Lab</groupf>
   <wwwg>http:// clip.dia.fi.upm.es</wwwg>
   <category>University</category>
   <description></description>
   <address>Campus MonteGancedo</address>
   <zip>28660</zip>
   <city>Boadilla del Monte, Madrid</city>
   <state></state>
   <country>Spain</country>
   <password>CLIP</password>
   <member><person>56</person></member>
   <contact_person><person>56</person></contact_person>
   <coordinating><project>62</project></coordinating>
   <related><software>0</software></related>
   <working>
    <area>0</area>
    <area>3</area>
    <area>2</area>
    <area>1</area>
   </working>
 </group>
</object>
<associated_data>
  <person>
    <identifier>56</identifier>
    <last_name>Cabeza</last_name>
    <first_name>Daniel</first_name>
    <password>bardo</password>
  </person>
  <project>
    <identifier>62</identifier>
    <titlea>CoLogNet</titlea>
    <titlef>CoLogNet</titlef>
    <password>colognet</password>
  </project>
  <software>
    <identifier>0</identifier>
    <namea>Ciao</namea>
    <namef>The Ciao Program Development System</namef>
    <password>ciao</password>
  </software>
  <area>
```

```
      <identifier>0</identifier>
      <namea>Automatic Deduction Systems</namea>
      <namef>Automated Deduction Systems and Theorem Provers</namef>
      <password>jorge</password>
    </area>
    <area>
      <identifier>3</identifier>
      <namea>Logic Programming</namea>
      <namef>Logic Programming</namef>
      <password>jorge</password>
    </area>
    <area>
      <identifier>2</identifier>
      <namea>Data Mining</namea>
      <namef>Data Mining</namef>
      <password>jorge</password>
    </area>
    <area>
      <identifier>1</identifier>
      <namea>CL Systems</namea>
      <namef>Computational Logic Systems</namef>
      <password>jorge</password>
    </area>
  </associated_data>
</register>
```

11.5 Example of Modification

```
    <modify>
  <object>
   <person>
     <identifier>56</identifier>
     <title>Mr</title>
     <www>http://clip.dia.fi.upm.es/~bardo</www>
     <email>bardo@clip.dia.fi.upm.es</email>
     <phone>123456789</phone>
     <fax></fax>
     <description></description>
     <address>Campus MonteGancedo</address>
     <zip>28660</zip>
     <city>Boadilla del Monte, Madrid</city>
     <state></state>
     <country>Spain</country>
     <old_last_name>Cabeza</old_last_name>
     <new_last_name>Cabeza</new_last_name>
     <old_first_name>Daniel</old_first_name>
     <new_first_name>Daniel</new_first_name>
     <old_password>bardo</old_password>
     <new_password>CLIP</new_password>
```

```
    <member><group>26</group></member>
    <contact_person></contact_person>
    <contact_software><software>0</software></contact_software>
   </person>
  </object>
  <associated_data>
   <group>
    <identifier>26</identifier>
    <groupa>CLIP</groupa>
    <groupf>CLIP Lab</groupf>
    <password>clip</password>
   </group>
   <software>
    <identifier>0</identifier>
    <namea>Ciao</namea>
    <namef>The Ciao Program Development System</namef>
    <password>ciao</password>
   </software>
  </associated_data>
 </modify>
```

11.6 Example of Deletion

```
   <unregister>
  <object>
    <group>
      <groupa>jorge</groupa>
      <groupf>jorge</groupf>
      <password>jorge</password>
    </group>
  </object>
  <associated_data> </associated_data>
 </unregister>
```

11.7 Version/Change Log (xsd)

Version 0.1#7 (2003/12/5, 12:47:36 CET)
Started automatic documentation (Jorge Navas)

12 Parsing, validating and retrieving of XML files

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#8 (2003/12/5, 12:47:52 CET)

This module allows to parser an XML file, to validate its content and to retrieve the data stored in it.

12.1 Usage and interface (parser_xml)

- **Library usage:**
  `:- use_module(library(parser_xml)).`
- **Exports:**
  - *Predicates:*
    `input_file_object_entity/4`, `input_file_associated_entity/3`, `process_file_entity/3`.
  - *Regular Types:*
    `op_xml/1`.
- **Other modules used:**
  - *Application modules:*
    `../../lib/pillow_ext/pillow_ext`, `../../src/entity/process_entity`, `../../src/structs`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

12.2 Documentation on exports (parser_xml)

input_file_object_entity/4: PREDICATE

`input_file_object_entity(Entity, Op, XML_Term, Input)`

Given an XML term (`XML_Term`) allows to parser the part corresponding to the `'object'` node, checking out that the parsed data matches with its corresponding XML schema, and retrieving that data in `Input`. The `'object'` node indicates the item corresponding to `Entity` on which is carried out the operation. The structure of the XML file is different depending on `Op`.

Usage 1: `input_file_object_entity(+Entity, +Op, +XML_Term, ?(Input))`

- *Description:* When `Op` is `'register'`, the root of the XML file is `'register'`. See Section 11.4 [Example of Insertion], page 66.
- *Call and exit should be compatible with:*
  `+Entity` is an entity. (`entity/1`)
  `+Op` and `register` unify. (`= /2`)
  `+XML_Term` is an XML file represented as a tree. (`xml_tree/1`)
  `?(Input)` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

Usage 2: `input_file_object_entity(+Entity, +Op, +XML_Tree, ?(Input))`

- *Description:* When `Op` is `'modify'`, the root of the XML file is `'modify'`. See Section 11.5 [Example of Modification], page 67.
- *Call and exit should be compatible with:*

  `+Entity` is an entity. (`entity/1`)

  `+Op` and `modify` unify. (`= /2`)

  `+XML_Tree` is an XML file represented as a tree. (`xml_tree/1`)

  `?(Input)` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

Usage 3: `input_file_object_entity(+Entity, +Op, +XML_Tree, ?(Input))`

- *Description:* When `Op` is `'unregister'`, the root of the XML file is `'unregister'`. See Section 11.6 [Example of Deletion], page 68.
- *Call and exit should be compatible with:*

  `+Entity` is an entity. (`entity/1`)

  `+Op` and `unregister` unify. (`= /2`)

  `+XML_Tree` is an XML file represented as a tree. (`xml_tree/1`)

  `?(Input)` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

input_file_associated_entity/3: PREDICATE

Usage: `input_file_associated_entity(+Entity, +XML_Term, ?(Input))`

- *Description:* Given an XML term (`XML_Term`) allows to parser the part corresponding to the `'associated_data'` node, checking out that the parsed data matches with its corresponding XML schema, and retrieving that data in `Input`. The `'associated_data'` node indicates the entities associated to the main entity (`Entity`) on which is carried out the operation.
- *Call and exit should be compatible with:*

  `+Entity` is an entity. (`entity/1`)

  `+XML_Term` is an XML file represented as a tree. (`xml_tree/1`)

  `?(Input)` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

process_file_entity/3: PREDICATE

Usage: `process_file_entity(+Entity, +Op, ?(Input))`

- *Description:* Once the XML file is parsed, validated, and the data corresponding to `Entity` has been retrieved in `Input`; this predicate inserts, modifies or deletes (depending on `Op`) that data in the system using the `process_entity/5` predicate.
- *Call and exit should be compatible with:*

  `+Entity` is an entity. (`entity/1`)

  `+Op` is the type of operation in an XML file. It can be: `'register'`, `'modify'` or `'unregister'`. (`op_xml/1`)

  `?(Input)` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`. (`input_entity_struct/1`)

op_xml/1: REGTYPE

Usage: `op_xml(Op)`

– *Description:* `Op` is the type of operation in an XML file. It can be: `'register'`, `'modify'` or `'unregister'`.

12.3 Version/Change Log (`parser_xml`)

Version 0.1#8 (2003/12/5, 12:47:52 CET)

Started automatic documentation (Jorge Navas)

13 XML code generation

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#9 (2003/12/5, 12:48:19 CET)

This module implements the necessary predicates for the creation of the XML code derived from a transaction produced over the database.

13.1 Usage and interface (code_xml)

- **Library usage:**
  `:- use_module(library(code_xml)).`
- **Exports:**
  - *Predicates:*
    `code_entities_xml/5`, `code_relations_xml/3`.
  - *Multifiles:*
    `$is_persistent/2`.
- **Other modules used:**
  - *Application modules:*
    `../../database/database`, `../../settings`, `./parser_xml`, `./xml_aux`, `./label_xml`, `../lists_ext/lists_ext`, `../date/date`, `../../src/structs`.
  - *System library modules:*
    `pillow/http`, `pillow/html`, `persdb/persdbrt`, `file_utils`, `terms`, `lists`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

13.2 Documentation on exports (code_xml)

code_entities_xml/5: PREDICATE

Usage: `code_entities_xml(+Entity, +Op, +Entity_Data, ?(List_Relations), -Object)`

- *Description:* `Object` is the XML code associated to the ``object'' node (see Chapter 11 [XSD associated to the XML data], page 49). This node represents the item on which the transaction has been carried out. The XML code is generated by interpreting the corresponding XML template, which is matched with two types of information: basic data of the item `Entity_Data` and the item data corresponding to the relationships with other items `List_Relations`. The choice of this template depends on the entity `Entity` and the type of operation `Op`.
- *Call and exit should be compatible with:*
  `+Entity` is an entity. (`entity/1`)
  `+Op` is the type of operation in an XML file. It can be: `'register'`, `'modify'` or `'unregister'`. (`op_xml/1`)

`+Entity_Data` is the union of the types: `output_area_struct/1`, `output_event_struct/1`, `output_group_struct/1`, `output_person_struct/1`, `output_project_struct/1` and `output_software_struct/1`. (`output_entity_struct/1`)

`?(List_Relations)` is a list of `ints`. (`list/2`)

`-Object` is an atom which represents the XML code associated to the main item. (`object_xml/1`)

code_relations_xml/3: PREDICATE

Usage: `code_relations_xml(+Entity, +Relations, -Associated)`

- *Description:* `Associated` is the XML code associated to the ``associated_data'' node (see Chapter 11 [XSD associated to the XML data], page 49). This node represents the items associated to the main item, corresponding to `Entity`, on which the transaction has been carried out. The XML code is generated by interpreting the corresponding XML template, which is matched with the item data corresponding to the relationships with other items `Relations`.
- *Call and exit should be compatible with:*

  `+Entity` is an entity. (`entity/1`)

  `+Relations` is a term that contains information about all relationships among the entities. (`relations_struct/1`)

  `-Associated` is a term that contains XML terms about the relationships among the different entities. (`associated_xml/1`)

13.3 Documentation on multifiles (`code_xml`)

$is_persistent/2: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

13.4 Documentation on internals (`code_xml`)

associated_xml/1: REGTYPE

`associated_xml(Associated)`

This type represents the relationships among the different entities. It is defined as:

```
associated_xml(relations(Member, Contact_Person, Contact_Software,
                         Coordinating, Related, Working)).
```

The arguments shown above mean:

- `Member` is a XML term that contains the groups' members.
- `Contact_Person` is a XML term contains the group's contact people.
- `Contact_Software` is a XML term contains the contact people of a software.

- `Coordinating` is a XML term contains the projects coordinated by the groups.
- `Related` is a XML term contains software related with the groups.
- `Working` is a XML term contains the areas related with the groups.

Usage: `associated_xml(Associated)`

– *Description:* `Associated` is a term that contains XML terms about the relationships among the different entities.

object_xml/1: REGTYPE

Usage: `object_xml(Object)`

– *Description:* `Object` is an atom which represents the XML code associated to the main item.

13.5 Version/Change Log (`code_xml`)

Version 0.1#9 (2003/12/5, 12:48:19 CET)

Started automatic documentation (Jorge Navas)

14 XML error processing

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#10 (2003/12/5, 12:48:41 CET)

This module implements a set of predicates to check out, to handle, and to notify errors in an XML file.

14.1 Usage and interface (error_xml)

- **Library usage:**
  `:- use_module(library(error_xml)).`
- **Exports:**
  - *Predicates:*
    `check_empty/1`, `check_entity_integrity/2`, `check_reference_integrity/2`, `check_error/1`, `get_template_error/3`, `notify_error_xml/1`.
- **Other modules used:**
  - *Application modules:*
    `./xml_aux`, `../../src/structs`, `../../settings`, `./code_xml`.
  - *System library modules:*
    `system`, `write`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

14.2 Documentation on exports (error_xml)

check_empty/1: PREDICATE

Usage: `check_empty(Val)`

- *Description:* Checks out that the value `Val` corresponding to a XML element is not empty.
- *Call and exit should be compatible with:*
  `Val` is an atom. (`atm/1`)

check_entity_integrity/2: PREDICATE

Usage: `check_entity_integrity(?(Id), ?(E))`

- *Description:* Checks out that the XML identifier `Id` corresponding to the entity `E` is unique.
- *Call and exit should be compatible with:*
  `?(Id)` is an XML identifier. (`id_xml/1`)
  `?(E)` is an entity. (`entity/1`)

check_reference_integrity/2: PREDICATE

Usage: `check_reference_integrity(+Ids, ?(E))`

- *Description:* Checks out that the referential integrity is fulfilled for a set of XML identifiers `Ids` corresponding to the entity `E`.
- *Call and exit should be compatible with:*

  `+Ids` is a list of `id_xmls`. (`list/2`)

  `?(E)` is an entity. (`entity/1`)

check_error/1: PREDICATE

Usage: `check_error(+Result_Code)`

- *Description:* Checks out that `Result_Code` corresponds to one of the different errors that can be produced in an XML file.
- *Call and exit should be compatible with:*

  `+Result_Code` is the result of processing an XML file. (`result_xml/1`)

get_template_error/3: PREDICATE

Usage: `get_template_error(+Result_Code, ?(Title), ?(Message))`

- *Description:* Given the result of processing an XML file (`Result_Code`), associates it to a title `Title` and a brief description `Message`.
- *Call and exit should be compatible with:*

  `+Result_Code` is the result of processing an XML file. (`result_xml/1`)

  `?(Title)` is an atom, which represents a title in HTML code, associated to the result of processing an XML file. (`title_html/1`)

  `?(Message)` is an atom, which represents a brief description in HTML code, associated to the result of processing an XML file. (`msg_html/1`)

notify_error_xml/1: PREDICATE

Usage: `notify_error_xml(+Error)`

- *Description:* Writes in a log file information about the error defined in `Error`.
- *Call and exit should be compatible with:*

  `+Error` is a term that contains information about the errors produced during the processing of an XML file. (`error_struct/1`)

14.3 Documentation on internals (`error_xml`)

result_xml/1: REGTYPE

Shows the result of processing an XML file. The different values can be:

- `'error_invalid_syntax'` states that the XML structure is not valid.
- `'error_entity_integrity'` states that there are repeated identifiers.
- `'error_reference_integrity'` states that the referential integrity is not carried out in the XML file.
- `'object_not_found'` states that an entity has not been found.

- `'error_empty'` states that there are empty required fields.
- `'error_category_event'` states that the event category is not correct.
- `'error_category_group'` states that the group category is not correct.
- `'error_date_incorrect'` states that a date is not correct.
- `'error_not_relations'` states that there are related entities.
- `'not_error'` states that the proccessing has been correct.

Usage: `result_xml(R)`

– *Description:* `R` is the result of processing an XML file.

title_html/1: REGTYPE

Usage: `title_html(T)`

– *Description:* `T` is an atom, which represents a title in HTML code, associated to the result of processing an XML file.

msg_html/1: REGTYPE

Usage: `msg_html(M)`

– *Description:* `M` is an atom, which represents a brief description in HTML code, associated to the result of processing an XML file.

14.4 Version/Change Log (`error_xml`)

Version 0.1#10 (2003/12/5, 12:48:41 CET)

Started automatic documentation (Jorge Navas)

15 XML file processing auxiliary predicates

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#11 (2003/12/5, 12:49:3 CET)

This library defines an interface for the `key_xml/2` and `key_pair_xml/3` predicates, where the first predicate allows handling the entity integrity and referential integrity in XML files, and the second predicate allows to define a equivalence between XML identifiers and database keys.

15.1 Usage and interface (`xml_aux`)

- **Library usage:**
  `:- use_module(library(xml_aux)).`
- **Exports:**
  - *Predicates:*
    `member_key/2`, `add_key/2`, `delete_key/2`, `deleteall_key/1`, `member_key_pair/3`, `add_key_pair/3`, `delete_key_pair/3`, `deleteall_key_pair/1`, `get_key_pair/3`.
  - *Regular Types:*
    `id_xml/1`.
- **Other modules used:**
  - *Application modules:*
    `../../src/entity/process_entity.pl`, `../../database/database`.
  - *System library modules:*
    `dynamic`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

15.2 Documentation on exports (`xml_aux`)

member_key/2: PREDICATE

Usage: `member_key(?(Id), ?(Entity))`

- *Description:* The XML identifier `Id` is associated to the entity `Entity`.
- *Call and exit should be compatible with:*
  `?(Id)` is an XML identifier. (`id_xml/1`)
  `?(Entity)` is an entity. (`entity/1`)

add_key/2: PREDICATE

Usage: `add_key(+Id, +Entity)`

- *Description:* Associates the XML identifier `Id` to the entity `Entity`.
- *Call and exit should be compatible with:*
  `+Id` is an XML identifier. (`id_xml/1`)
  `+Entity` is an entity. (`entity/1`)

delete_key/2: PREDICATE

Usage: `delete_key(+Id, +Entity)`

- *Description:* Deletes the relation between the XML identifier `Id` and the entity `Entity`.
- *Call and exit should be compatible with:*

  `+Id` is an XML identifier. (`id_xml/1`)

  `+Entity` is an entity. (`entity/1`)

deleteall_key/1: PREDICATE

Usage: `deleteall_key(+Entity)`

- *Description:* Deletes the relation between the XML identifiers and the entity `Entity`.
- *Call and exit should be compatible with:*

  `+Entity` is an entity. (`entity/1`)

member_key_pair/3: PREDICATE

Usage: `member_key_pair(?(Id), ?(Key), ?(Entity))`

- *Description:* There is a equivalence between the XML identifier `Id`, associated to the entity `Entity`, and the internal key `Key`.
- *Call and exit should be compatible with:*

  `?(Id)` is an XML identifier. (`id_xml/1`)

  `?(Key)` is an integer which represents a database key. (`key_db_int/1`)

  `?(Entity)` is an entity. (`entity/1`)

add_key_pair/3: PREDICATE

Usage: `add_key_pair(?(Id), ?(Key), ?(Entity))`

- *Description:* Associates the XML identifier `Id`, corresponding to the entity `Entity`, to the internal key `Key`.
- *Call and exit should be compatible with:*

  `?(Id)` is an XML identifier. (`id_xml/1`)

  `?(Key)` is an integer which represents a database key. (`key_db_int/1`)

  `?(Entity)` is an entity. (`entity/1`)

delete_key_pair/3: PREDICATE

Usage: `delete_key_pair(?(Id), ?(Key), ?(Entity))`

- *Description:* Deletes the equivalence between the XML identifier `Id`, associated to the entity `Entity`, and the internal key `Key`.
- *Call and exit should be compatible with:*

  `?(Id)` is an XML identifier. (`id_xml/1`)

  `?(Key)` is an integer which represents a database key. (`key_db_int/1`)

  `?(Entity)` is an entity. (`entity/1`)

deleteall_key_pair/1: PREDICATE

Usage: `deleteall_key_pair(?(Entity))`

- *Description:* Deletes all the equivalences between all the XML identifiers associated to the entity `Entity`, and their corresponding internal keys.
- *Call and exit should be compatible with:*

  `?(Entity)` is an entity. (`entity/1`)

get_key_pair/3: PREDICATE

Usage: `get_key_pair(+Ids, +Entity, ?(Keys))`

- *Description:* `Keys` is a list of internal keys, which are provided by the XML identifiers `Ids` associated to the entity `Entity`.
- *Call and exit should be compatible with:*

  `+Ids` is a list of `id_xml`s. (`list/2`)

  `+Entity` is an entity. (`entity/1`)

  `?(Keys)` is a list of `key_db_int`s. (`list/2`)

id_xml/1: REGTYPE

Usage: `id_xml(Id)`

- *Description:* `Id` is an XML identifier.

15.3 Documentation on internals (`xml_aux`)

key_xml/2: PREDICATE

The predicate is of type *dynamic*.

Usage: `key_xml(Id, Entity)`

- *Description:* `Id` is associated to entity `Entity`.
- *Call and exit should be compatible with:*

  `Id` is an XML identifier. (`id_xml/1`)

  `Entity` is an entity. (`entity/1`)

key_pair_xml/3: PREDICATE

The predicate is of type *dynamic*.

Usage: `key_pair_xml(Id, Key, Entity)`

- *Description:* `Id` is associated to internal key `Key` and to the entity `Entity`.
- *Call and exit should be compatible with:*

  `Id` is an XML identifier. (`id_xml/1`)

  `Key` is an integer which represents a database key. (`key_db_int/1`)

  `Entity` is an entity. (`entity/1`)

15.4 Version/Change Log (`xml_aux`)

Version 0.1#11 (2003/12/5, 12:49:3 CET)

Started automatic documentation (Jorge Navas)

16 XML labels

Version: 0.1#27 (2003/12/5, 13:14:2 CET)
Version of last change: 0.1#12 (2003/12/5, 12:49:27 CET)
This library implements a set of predicates for the generation of XML labels.

16.1 Usage and interface (`label_xml`)

- **Library usage:**
  `:- use_module(library(label_xml)).`
- **Exports:**
  - *Predicates:*
    `gen_label_body/3`, `gen_label_init/2`, `gen_label_end/2`.
- **Other modules used:**
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

16.2 Documentation on exports (`label_xml`)

gen_label_body/3: PREDICATE

Usage: `gen_label_body(+Name, +Body, ?(Tag))`

- *Description:* Generates a XML label `Tag` whose name is `Name` and whose body is `Body`.
- *Call and exit should be compatible with:*
  `+Name` is an atom. (`atm/1`)
  `+Body` is an atom. (`atm/1`)
  `?(Tag)` is an atom. (`atm/1`)

gen_label_init/2: PREDICATE

Usage: `gen_label_init(+Name, ?(Tag_Init))`

- *Description:* Generates a start XML label `Tag_Init` whose name is `Name`.
- *Call and exit should be compatible with:*
  `+Name` is an atom. (`atm/1`)
  `?(Tag_Init)` is an atom. (`atm/1`)

gen_label_end/2: PREDICATE

Usage: `gen_label_end(+Name, ?(Tag_End))`

- *Description:* Generates a end XML label `Tag_End` whose name is `Name`.
- *Call and exit should be compatible with:*
  `+Name` is an atom. (`atm/1`)
  `?(Tag_End)` is an atom. (`atm/1`)

16.3 Version/Change Log (`label_xml`)

Version 0.1#12 (2003/12/5, 12:49:27 CET)
Started automatic documentation (Jorge Navas)

17 Output from data exchange status

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#13 (2003/12/5, 12:50:5 CET)

This CGI application outputs a HTML page with the content of a log file, showing all errors produced during the data exchange among the Web sites, to make easier the administration tasks.

17.1 Usage and interface (errors_log)

- **Library usage:**
  This module is typically compiled as a CGI executable.
- **Other modules used:**
  - *Application modules:*
    `../lib/xml/error_xml`, `../settings`, `../lib/date/date`.
  - *System library modules:*
    `pillow/http`, `pillow/html`, `file_utils`, `sort`, `lists`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

17.2 Documentation on internals (errors_log)

main/0: PREDICATE

Usage:

- *Description:* Entry predicate to CGI executable.

17.3 Version/Change Log (errors_log)

Version 0.1#13 (2003/12/5, 12:50:5 CET)

Started automatic documentation (Jorge Navas)

18 Structures related to input/output data

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#14 (2003/12/5, 12:50:25 CET)

This module contains a set of basic structures to deal with input/output data. These input operations can be from an HTML form or an XML file. The output operations can be either to the own Web site by the CGI protocol and HTML code or to other web sites by the HTTP protocol and XML files.

18.1 Usage and interface (structs)

- **Library usage:**
  `:- use_module(library(structs)).`
- **Exports:**
  - *Regular Types:*
    `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1`, `input_software_struct/1`, `input_event_struct/1`, `input_area_struct/1`, `input_file_struct/1`, `input_entity_struct/1`, `output_group_struct/1`, `output_person_struct/1`, `output_project_struct/1`, `output_software_struct/1`, `output_event_struct/1`, `output_area_struct/1`, `output_entity_struct/1`, `relations_struct/1`, `error_struct/1`.
- **Other modules used:**
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

18.2 Documentation on exports (structs)

input_group_struct/1: REGTYPE

`input_group_struct(Input_Group_Struct)`

This type represents group data obtained by input operations. It is defined as:

```
input_group_struct(input_group(Key,Password, Inst_a,Inst_f,WWW_inst,
                               Group_a,Group_f,WWW_group,
                               Category,Description,
                               Address,Zip,State,City,Country,
                               Member,Contact,Coordinating,
                               Related,Working,Subop,Id,
                               Old_Group_a,Old_Group_f,Old_Password,
                               Id_Session)).
```

The arguments shown above mean:

- `Key` is a unique identifier.

- `Password` is an access key.
- `Inst_a` is the institution's abbreviated name to which the group belongs to.
- `Inst_f` is the institution's complete name to which the group belongs to.
- `WWW_inst` is the institution's Web address to which the group belongs to.
- `Group_a` is the abbreviated name.
- `Group_f` is the complete name.
- `WWW_group` is the Web address.
- `Category` can be: `'company'`, `'non profit research institute'`, `'research institute'` or `'university'`.
- `Description` is a brief description.
- `Address` is the address.
- `Zip` is the zip code.
- `State` is the state (it is optional).
- `City` is the city.
- `Country` is the country.
- `Member` are the people belonging to the group.
- `Contact` are the group's contact people.
- `Coordinating` are the projects coordinated by the group.
- `Related` is software related with the group.
- `Working` are the areas associated to the group.
- `Subop` can be `'modify'` or `'delete'`.
- `Id` is the value of the element `'identifier'` in an XML file.
- `Old_Group_a` is the abbreviated name before carrying out the modification of group data.
- `Old_Group_f` is the complete name before carrying out the modification of group data.
- `Old_Password` is the access key before carrying out the modification of group data.
- `Id_Session` is a session identifier.

Usage: `input_group_struct(Input_Group_Struct)`

– *Description:* `Input_Group_Struct` is a term that contains group data, obtained by an input operation.

input_person_struct/1: REGTYPE

`input_person_struct(Input_Person_Struct)`

This type represents person data obtained by input operations. It is defined as:

```
input_person_struct(input_person(Key,Password, Last_Name,First_Name,
                                 Title,WWW,Email,Phone,Fax,
                                 Description,
                                 Address,Zip,State,Country,
                                 Member,Contact_Group,Contact_Soft,
                                 Subop,Id,
                                 Old_Last_Name,Old_First_Name,
                                 Old_Password, Id_Session)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Last_Name` is the last name.
- `First_Name` is the first name.
- `Title` is the degree.
- `WWW` is the Web address.
- `Email` is the email.
- `Phone` is the telephone number.
- `Fax` is the fax number.
- `Description` is a brief description.
- `Address` is the address.
- `Zip` is the zip code.
- `State` is the state where the researcher lives. (It is optional).
- `City` is the city where the person lives.
- `Country` is the country where the person lives.
- `Member` are the groups of which the person is member.
- `Contact_Group` are the groups of which the person is contact.
- `Contact_Soft` is the software of which the person is contact.
- `Subop` can be: `'modify'` or `'delete'`.
- `Id` is the value of the element `'identifier'` in an XML file.
- `Old_Last_Name` is the last name before carrying out the modification of person data.
- `Old_First_Name` is the first name before carrying out the modification of person data.
- `Old_Password` is the access key before carrying out the modification of person data.
- `Id_Session` is a session identifier.

Usage: `input_person_struct(Input_Person_Struct)`

- *Description:* `Input_Person_Struct` is a term that contains person data, obtained by an input operation.

input_project_struct/1: REGTYPE

`input_project_struct(Input_Project_Struct)`

This type represents project data obtained by input operations. It is defined as:

```
input_project_struct(input_project(Key,Password, Title_a,Title_f,WWW,
                                   Category,Description,Comment,
                                   Coordinating, Subop,Id,
                                   Old_Title_a,Old_Title_f,
                                   Old_Password,
                                   Id_Session)).
```

The arguments shown above mean:

- `Key` is a unique identifier.

- `Password` is an access key.
- `Title_a` is the abbreviated title.
- `Title_f` is the complete title.
- `WWW` is the Web address.
- `Category` is the category.
- `Description` is a brief description.
- `Comment` is an additional comment.
- `Coordinating` are the groups which coordinate the project.
- `Subop` can be: `'modify'` or `'delete'`.
- `Id` is the value of the element `'identifier'` in a XML file.
- `Old_Title_a` is the abbreviated title before carrying out the modification of project data.
- `Old_Title_f` is the complete title before carrying out the modification of project data.
- `Old_Password` is the access key before carrying out the modification of project data.
- `Id_Session` is a session identifier.

Usage: `input_project_struct(Input_Project_Struct)`

– *Description:* `Input_Project_Struct` is a term that contains project data, obtained by an input operation.

input_software_struct/1: REGTYPE

`input_software_struct(Input_Software_Struct)`

This type represents software data obtained by input operations. It is defined as:

```
input_software_struct(input_software(Key,Password, Name_a,Name_f,
                                     WWW,FTP,
                                     Platforms,Availability,
                                     Description,
                                     Related,Contact,
                                     Subop,Id,
                                     Old_Name_a,Old_Name_f,
                                     Old_Password,
                                     Id_Session)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Name_a` is the abbreviated name.
- `Name_f` is the complete name.
- `WWW` is the Web address.
- `FTP` is the FTP address.
- `Platforms` are the platforms supported.
- `Availability` is the type of license.
- `Description` is a brief description.

- `Related` are groups related with the software.
- `Contact` are contact people.
- `Subop` can be: `'modify'` or `'delete'`.
- `Id` is the value of the field `'identifier'` in a XML file.
- `Old_Name_a` is the abbreviated name before carrying out the modification of software data.
- `Old_Name_f` is the complete name before carrying out the modification of software data.
- `Old_Password` is the access key before carrying out the modification of software data.
- `Id_Session` is a session identifier.

Usage: `input_software_struct(Input_Software_Struct)`

– *Description:* `Input_Software_Struct` is a term that contains software data, obtained by an input operation.

input_event_struct/1: REGTYPE

`input_event_struct(Input_Event_Struct)`

This type represents event data obtained by input operations. It is defined as:

```
input_event_struct(input_event(Key,Password, Title,
                               Day_f,Month_f,Year_f,
                               Day_l,Month_l,Year_l,
                               Day_d,Month_d,Year_d,
                               Category,City,Country,Comment,
                               Person_Contact,Email, Subop,
                               Old_Title,Old_Password, Id_Session)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Title` is the title.
- `Day_f` is the start day.
- `Month_f` is the start month.
- `Year_f` is the start year.
- `Day_l` is the end day.
- `Month_l` is the end month.
- `Year_l` is the end year.
- `Day_d` is the deadline day.
- `Month_d` is the deadline month.
- `Year_d` is the deadline year.
- `Category` is the category.
- `City` is the city where the event is held.
- `Country` is the country where the event is held.
- `Comment` is an additional comment.

- `Person_Contact` is the contact person.
- `Email` is the contact person's email.
- `Subop` can be: `'modify'` or `'delete'`.
- `Id` is the value of the field `'identifier'` in an XML file.
- `Old_Title` is the title before carrying out the modification of event data.
- `Old_Password` is the access key before carrying out the modification of event data.
- `Id_Session` is a session identifier.

Usage: `input_event_struct(Input_Event_Struct)`

– *Description:* `Input_Event_Struct` is a term that contains event data, obtained by an input operation.

input_area_struct/1: REGTYPE

`input_area_struct(Input_Area_Struct)`

This type represents area data obtained by input operations. It is defined as:

```
input_area_struct(input_area(Key,Password, Name_a,Name_f,
                             Description,
                             Working, Subop,Id,
                             Old_Name_a,Old_Name_f,Old_Password,
                             Id_Session)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Name_a` is the abbreviated name.
- `Name_f` is the complete name.
- `Description` is a brief description.
- `Working` are the groups that are classified in the area.
- `Subop` can be: `'modify'` or `'delete'`.
- `Id` is the value of the field *'identifier'* in an XML file.
- `Old_Name_a` is the abbreviated name before carrying out the modification of area data.
- `Old_Name_f` is the complete name before carrying out the modification of area data.
- `Old_Password` is the access key before carrying out the modification of area data.
- `Id_Session` is a session identifier.

Usage: `input_area_struct(Input_Area_Struct)`

– *Description:* `Input_Area_Struct` is a term that contains area data, obtained by an input operation.

input_file_struct/1: REGTYPE

`input_file_struct(Input_File_Struct)`

This type represents the content of a file. It is defined as:

```
input_file_struct(input_file(File)).
```

The arguments shown above mean:

- `File` is the content of an XML file.

Usage: `input_file_struct(Input_File_Struct)`

- *Description:* `Input_File_Struct` is a term that stores the content of a file.

input_entity_struct/1: REGTYPE

Usage: `input_entity_struct(Input_Entity_Struct)`

- *Description:* `Input_Entity_Struct` is the union of the types: `input_area_struct/1`, `input_event_struct/1`, `input_group_struct/1`, `input_person_struct/1`, `input_project_struct/1` and `input_software_struct/1`.

output_group_struct/1: REGTYPE

`output_group_struct(Output_Group_Struct)`

This type represents data about a group, used for output operations. It is defined as:

```
output_group_struct(output_group(Key,Password,
                                 Inst_a,Inst_f,WWW_inst,
                                 Group_a,Group_f,WWW_group,
                                 Category,Description,
                                 Address,Zip,State,City,
                                 Country,Member,Contact,
                                 Coordinating,Related,Working,
                                 Title_filen,
                                 Old_Group_a,Old_Group_f,
 Old_Password)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Inst_a` is the institution's abbreviated name to which the group belongs to.
- `Inst_f` is the institution's complete name to which the group belongs to.
- `WWW_inst` is the institution's Web address to which the group belongs to.
- `Group_a` is the abbreviated name.
- `Group_f` is the complete name.
- `WWW_group` is the group's Web address.
- `Category` is the category.
- `Description` is a brief description.
- `Address` is the address.
- `Zip` is the zip code.

- `State` is the state (it is optional).
- `City` is the city.
- `Country` is the country.
- `Member` are the researchers of the group.
- `Contact` are the contact people.
- `Coordinating` are the projects coordinated by the group.
- `Related` is software related with the group.
- `Working` are the areas related with the group.
- `Title_file` is a message, used for HTML output, when an operation is carried out in several steps.
- `Old_Group_a` is the abbreviated name before carrying out the modification of group data.
- `Old_Group_f` is the complete name before carrying out the modification of group data.
- `Old_Password` is the access key before carrying out the modification of group data.

Usage: `output_group_struct(Output_Group_Struct)`

– *Description:* `Output_Group_Struct` is a term that contains data about a group, used for output operations.

output_person_struct/1: REGTYPE

`output_person_struct(Output_Person_Struct)`

This type represents data about a person, used for output operations. It is defined as:

```
output_person_struct(output_person(Key,Password,
                                   Last_Name,First_Name,Title,
                                   WWW,Email,Phone,Fax,Description,
                                   Address,Zip,State,City,Country,
                                   Member,Contact_Group,Contact_Soft,
                                   Title_file,
                                   Old_Last_Name,Old_First_Name,
                                   Old_Password)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Last_Name` is the last name.
- `First_Name` is the first name.
- `Title` is the degree.
- `WWW` is the Web address.
- `Email` is the email.
- `Phone` is the telephone number.
- `Fax` is the fax number.
- `Description` is a brief description.
- `Address` is the address.

- `Zip` is the zip code.
- `State` is the state where the person lives. (It is optional).
- `City` is the city where the person lives.
- `Country` is the country where the person lives.
- `Member` are the groups to which the person belongs to.
- `Contact_Group` are the groups of which the person is contact.
- `Contact_Soft` is the software of which the person is contact.
- `Title_file` is a message, used for HTML output, when an operation is carried out in several steps.
- `Old_Last_Name` is the last name before carrying out the modification of person data.
- `Old_First_Name` is the first name before carrying out the modification of person data.
- `Old_Password` is the access key before carrying out the modification of person data.

Usage: `output_person_struct(Output_Person_Struct)`

– *Description:* `Output_Person_Struct` is a term that contains data about a person, used for output operations.

output_project_struct/1: REGTYPE

`output_project_struct(Output_Project_Struct)`

This type represents data about a project, used for output operations. It is defined as:

```
output_project_struct(output_project(Key,Password, Title_a,Title_f,
                                     WWW,Category,
                                     Description,Comment,
                                     Coordinating,Title,
                                     Old_Title_a,Old_Title_f,
                                     Old_Password)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Title_a` is the abbreviated title.
- `Title_f` is the complete title.
- `WWW` is the Web address.
- `Category` is the category.
- `Description` is a brief description.
- `Comment` is an additional comment.
- `Coordinating` are the groups which coordinate the project.
- `Title` is a message, used for HTML output, when an operation is carried out in several steps.
- `Old_Title_a` is the abbreviated title before carrying out the modification of project data.
- `Old_Title_f` is the complete title before carrying out the modification of project data.

- `Old_Password` is the access key before carrying out the modification of project data.

Usage: `output_project_struct(Output_Project_Struct)`

– *Description:* `Output_Project_Struct` is a term that contains data about a project, used for output operations.

output_software_struct/1: REGTYPE

`output_software_struct(Output_Software_Struct)`

This type represents data about a software, used for output operations. It is defined as:

```
output_software_struct(output_software(Key,Password, Name_a,Name_f,
                                       WWW,FTP,
                                       Platforms,Availability,
                                       Description,
                                       Related,Contact,Title,
                                       Old_Name_a,Old_Name_f,
                                       Old_Password)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Name_a` is the abbreviated name.
- `Name_f` is the complete name.
- `WWW` is the Web address.
- `FTP` is the FTP address.
- `Platforms` are the platforms supported.
- `Availability` is the type of license.
- `Description` is a brief description.
- `Related` are groups related with the software.
- `Contact` are contact people.
- `Title` is a message, used for HTML output, when an operation is carried out in several steps.
- `Old_Name_a` is the abbreviated name before carrying out the modification of software data.
- `Old_Name_f` is the complete name before carrying out the modification of software data.
- `Old_Password` is the access key before carrying out the modification of software data.

Usage: `output_software_struct(Output_Software_Struct)`

– *Description:* `Output_Software_Struct` is a term that contains data about a software, used for output operations.

output_event_struct/1: REGTYPE

`output_event_struct(Output_Event_Struct)`

This type represents data about an event, used for output operations. It is defined as:

```
output_event_struct(output_event(Password, City,Country,
                                 Category,Title,
                                 Day_f,Month_f,Year_f,
                                 Day_l,Month_l,Year_l,
                                 Day_d,Month_d,Year_d, Comment,
                                 Person_Contact,Email, Title_f,
                                 Old_Title,Old_Password)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Title` is the title.
- `Day_f` is the start day.
- `Month_f` is the start month.
- `Year_f` is the start year.
- `Day_l` is the end day.
- `Month_l` is the end month.
- `Year_l` is the end year.
- `Day_d` is the deadline day.
- `Month_d` is the deadline month.
- `Year_d` is the deadline year.
- `Category` is the category of the event.
- `City` is the city where the event is held.
- `Country` is the country where the event is held.
- `Comment` is an additional comment.
- `Person_Contact` is the contact person.
- `Email` is the contact person's email.
- `Title_f` is a message, used for HTML output, when an operation is carried out in several steps.
- `Old_Title` is the title before carrying out the modification of event data.
- `Old_Password` is the access key before carrying out the modification of event data.

Usage: `output_event_struct(Output_Event_Struct)`

– *Description:* `Output_Event_Struct` is a term that contains data about an event, used for output operations.

output_area_struct/1: REGTYPE

`output_area_struct(Output_Area_Struct)`

This type represents data about an area, used for output operations. It is defined as:

```
output_area_struct(output_area(Key,Password, Name_a,Name_f,
                               Description,Title,
                               Old_Name_a,Old_Name_f,Old_Password)).
```

The arguments shown above mean:

- `Key` is a unique identifier.
- `Password` is an access key.
- `Name_a` is the abbreviated name.
- `Name_f` is the complete name.
- `Description` is a brief description.
- `Title_f` is a message, used for HTML output, when an operation is carried out in several steps.
- `Old_Name_a` is the abbreviated name before carrying out the modification of area data.
- `Old_Name_f` is the complete name before carrying out the modification of area data.
- `Old_Password` is the access key before carrying out the modification of area data.

Usage: `output_area_struct(Output_Area_Struct)`

– *Description:* `Output_Area_Struct` is a term that contains data about an area, used for output operations.

output_entity_struct/1: REGTYPE

Usage: `output_entity_struct(Output_Entity_Struct)`

– *Description:* `Output_Entity_Struct` is the union of the types: `output_area_struct/1`, `output_event_struct/1`, `output_group_struct/1`, `output_person_struct/1`, `output_project_struct/1` and `output_software_struct/1`.

relations_struct/1: REGTYPE

`relations_struct(Relations_Struct)`

This type represents the relationships among the entities. Depending on each entity, differents relationships can be used. It is defined as:

```
relations_struct(relations(Member, Contact_Person, Contact_Software,
                           Coordinating, Related, Working)).
```

The arguments shown above mean:

- `Member` contains the groups' members.
- `Contact_Person` contains the group's contact people.
- `Contact_Software` contains the contact people of a software.
- `Coordinating` contains the projects coordinated by the groups.
- `Related` contains software related with the groups.
- `Working` contains the areas related with the groups.

Usage: `relations_struct(Relations_Struct)`

– *Description:* `Relations_Struct` is a term that contains information about all relationships among the entities.

error_struct/1: REGTYPE

`error_struct(Relations_Struct)`

This type represents the errors produced during the processing of an XML file. Its defined as:

```
error_struct(error(Date,Type,File,Op)).
```

The arguments shown above mean:

- `Date` is the date when the error was produced.
- `Type` is the type of error.
- `File` is the complete path of the erroneous XML file.
- `Op` states if the error was produced in a sending or receiving.

Usage: `error_struct(Error_Struct)`

– *Description:* `Error_Struct` is a term that contains information about the errors produced during the processing of an XML file.

18.3 Version/Change Log (`structs`)

Version 0.1#14 (2003/12/5, 12:50:25 CET)

Started automatic documentation (Jorge Navas)

PART III - Database

This part includes a number of libraries that allow defining and manipulating the application database.

19 Database persistent predicates

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#15 (2003/12/5, 12:50:52 CET)

This module defines the application database. The database is implemented by *persistent predicates*.

19.1 Usage and interface (database)

- **Library usage:**
  `:- use_module(library(database)).`
- **Exports:**
  - *Predicates:*
    `group/1`, `person/1`, `project/1`, `software/1`, `event/1`, `area/1`, `member_of/2`, `working/2`, `contact_person_group/2`, `coordinating_group/2`, `related_group/2`, `contact_person_soft/2`.
  - *Regular Types:*
    `key_db_atm/1`, `key_db_int/1`.
  - *Multifiles:*
    `$is_persistent/2`.
- **Other modules used:**
  - *Application modules:*
    `../src/types`.
  - *System library modules:*
    `persdb/persdbrt`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

19.2 Documentation on exports (database)

group/1: PREDICATE

Contains data about a group.

The predicate is of type *data*.

Usage: `group(?(G))`

- *Description:* `G` is data about a group.
- *Call and exit should be compatible with:*
  `?(G)` is a term that contains data about a group. (`group_struct/1`)

person/1: PREDICATE

Contains data about a person.

The predicate is of type *data*.

Usage: `person(?(Per))`

- *Description:* `Per` is data about a person.
- *Call and exit should be compatible with:*

  `?(Per)` is a term that contains data about a person. (`person_struct/1`)

project/1: PREDICATE

Contains data about a project.

The predicate is of type *data*.

Usage: `project(?(Pro))`

- *Description:* `Pro` is data about a project.
- *Call and exit should be compatible with:*

  `?(Pro)` is a term that contains data about a project. (`project_struct/1`)

software/1: PREDICATE

Contains data about a software.

The predicate is of type *data*.

Usage: `software(?(S))`

- *Description:* `S` is data about a software.
- *Call and exit should be compatible with:*

  `?(S)` is a term that contains data about a software. (`software_struct/1`)

event/1: PREDICATE

Contains data about an event.

The predicate is of type *data*.

Usage: `event(?(E))`

- *Description:* `E` is data about an event.
- *Call and exit should be compatible with:*

  `?(E)` is a term that contains data about an event. (`event_struct/1`)

area/1: PREDICATE

Contains data about an area.

The predicate is of type *data*.

Usage: `area(?(A))`

- *Description:* `A` is data about an area.
- *Call and exit should be compatible with:*

  `?(A)` is a term that contains data about an area. (`area_struct/1`)

member_of/2: PREDICATE

The predicate is of type *data*.

Usage: `member_of(?(G), ?(Per))`

- *Description:* `Per` is a person belonging to the group `G`.
- *Call and exit should be compatible with:*

  `?(G)` is an atom which represents a database key. (`key_db_atm/1`)

  `?(Per)` is an atom which represents a database key. (`key_db_atm/1`)

working/2: PREDICATE

The predicate is of type *data*.

Usage: `working(?(G), ?(A))`

- *Description:* `G` is a group classified in the area `A`.
- *Call and exit should be compatible with:*

  `?(G)` is an atom which represents a database key. (`key_db_atm/1`)

  `?(A)` is an atom which represents a database key. (`key_db_atm/1`)

contact_person_group/2: PREDICATE

The predicate is of type *data*.

Usage: `contact_person_group(?(G), ?(Per))`

- *Description:* `Per` is a contact person of the group `G`.
- *Call and exit should be compatible with:*

  `?(G)` is an atom which represents a database key. (`key_db_atm/1`)

  `?(Per)` is an atom which represents a database key. (`key_db_atm/1`)

coordinating_group/2: PREDICATE

The predicate is of type *data*.

Usage: `coordinating_group(?(G), ?(Pro))`

- *Description:* `Pro` is a project coordinated by the group `G`.
- *Call and exit should be compatible with:*

  `?(G)` is an atom which represents a database key. (`key_db_atm/1`)

  `?(Pro)` is an atom which represents a database key. (`key_db_atm/1`)

related_group/2: PREDICATE

The predicate is of type *data*.

Usage: `related_group(?(G), ?(S))`

- *Description:* `S` is a software related with the group `G`.
- *Call and exit should be compatible with:*

  `?(G)` is an atom which represents a database key. (`key_db_atm/1`)

  `?(S)` is an atom which represents a database key. (`key_db_atm/1`)

contact_person_soft/2: PREDICATE

The predicate is of type *data*.

Usage: `contact_person_soft(?(Per), ?(S))`

- *Description:* `Per` is a contact person of the software `S`.
- *Call and exit should be compatible with:*

  `?(Per)` is an atom which represents a database key. (`key_db_atm/1`)

  `?(S)` is an atom which represents a database key. (`key_db_atm/1`)

key_db_atm/1: REGTYPE

Usage: `key_db_atm(Key)`

- *Description:* `Key` is an atom which represents a database key.

key_db_int/1: REGTYPE

Usage: `key_db_int(Key)`

- *Description:* `Key` is an integer which represents a database key.

19.3 Documentation on multifiles (database)

$is_persistent/2: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

19.4 Version/Change Log (database)

Version 0.1#15 (2003/12/5, 12:50:52 CET)

Started automatic documentation (Jorge Navas)

20 Database queries

Version: 0.1#27 (2003/12/5, 13:14:2 CET)
Version of last change: 0.1#16 (2003/12/5, 12:51:18 CET)
This module defines a set of predicates to carry out predefined queries over the database.

20.1 Usage and interface (queries_db)

- **Library usage:**
  `:- use_module(library(queries_db)).`
- **Exports:**
  - *Predicates:*
    `all_groups/1`, `all_people/1`, `all_projects/1`, `all_software/1`, `all_areas/1`, `working/4`, `member_of/4`, `coordinating_group/4`, `contact_person_group/4`, `related_group/4`, `contact_person_soft/4`.
  - *Multifiles:*
    `$is_persistent/2`.
- **Other modules used:**
  - *Application modules:*
    `../../database/database`, `../../lib/pillow_ext/pillow_ext`.
  - *System library modules:*
    `persdb/persdbrt`, `aggregates`, `terms`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

20.2 Documentation on exports (queries_db)

all_groups/1: PREDICATE

Usage: `all_groups(?(Gs))`

- *Description:* `Gs` are all the groups stored in the database.
- *Call and exit should be compatible with:*
  `?(Gs)` is a list of `group_reds`. (`list/2`)

all_people/1: PREDICATE

Usage: `all_people(?(Ps))`

- *Description:* `Ps` are all the people stored in the database.
- *Call and exit should be compatible with:*
  `?(Ps)` is a list of `person_reds`. (`list/2`)

all_projects/1: PREDICATE

Usage: `all_projects(?(Pros))`

- *Description:* `Pros` are all the projects stored in the database.
- *Call and exit should be compatible with:*

  `?(Pros)` is a list of `project_reds`. (`list/2`)

all_software/1: PREDICATE

Usage: `all_software(?(Ss))`

- *Description:* `Ss` is all the software stored in the database.
- *Call and exit should be compatible with:*

  `?(Ss)` is a list of `software_reds`. (`list/2`)

all_areas/1: PREDICATE

Usage: `all_areas(?(As))`

- *Description:* `As` are all the areas stored in the database.
- *Call and exit should be compatible with:*

  `?(As)` is a list of `area_reds`. (`list/2`)

working/4: PREDICATE

Usage 1: `working(+Entity, +Key_Area, ?(URL), ?(G))`

- *Description:* When `Entity` is `'group'`, `G` is the list of groups classified in the area `Key_Area`. Specifically, if `URL` is non ground, `G` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.
- *Call and exit should be compatible with:*

  `+Entity` and `group` unify. (`= /2`)

  `+Key_Area` is an atom which represents a database key. (`key_db_atm/1`)

  `?(URL)` is an URL (Uniform Resource Locator). (`url/1`)

  `?(G)` is a list of `terms`. (`list/2`)

Usage 2: `working(+Entity, +Key_Group, ?(URL), ?(A))`

- *Description:* When `Entity` is `'area'`, `A` is the list of areas to which the group `Key_Group` belongs. Specifically, if `URL` is non ground, `A` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.
- *Call and exit should be compatible with:*

  `+Entity` and `area` unify. (`= /2`)

  `+Key_Group` is an atom which represents a database key. (`key_db_atm/1`)

  `?(URL)` is an URL (Uniform Resource Locator). (`url/1`)

  `?(A)` is a list of `terms`. (`list/2`)

member_of/4: PREDICATE

Usage 1: `member_of(+Entity, +Key_Group, ?(URL), ?(P))`

– *Description:* When `Entity` is `'person'`, `P` is the list of people belonging to the group `Key_Group`. Specifically, if `URL` is non ground, `P` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.

– *Call and exit should be compatible with:*

`+Entity` and `person` unify.	`(= /2)`
`+Key_Group` is an atom which represents a database key.	`(key_db_atm/1)`
`?(URL)` is an URL (Uniform Resource Locator).	`(url/1)`
`?(P)` is a list of `terms`.	`(list/2)`

Usage 2: `member_of(+Entity, +Key_Person, ?(URL), ?(G))`

– *Description:* When `Entity` is `'group'`, `G` is the list of groups to which the person `Key_Person` belongs. Specifically, if `URL` is non ground, `G` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.

– *Call and exit should be compatible with:*

`+Entity` and `group` unify.	`(= /2)`
`+Key_Person` is an atom which represents a database key.	`(key_db_atm/1)`
`?(URL)` is an URL (Uniform Resource Locator).	`(url/1)`
`?(G)` is a list of `terms`.	`(list/2)`

coordinating_group/4: PREDICATE

Usage 1: `coordinating_group(+Entity, +Key_Group, ?(URL), ?(P))`

– *Description:* When `Entity` is `'project'`, `P` is the list of projects coordinated by the group `Key_Group`. Specifically, if `URL` is non ground, `P` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.

– *Call and exit should be compatible with:*

`+Entity` and `project` unify.	`(= /2)`
`+Key_Group` is an atom which represents a database key.	`(key_db_atm/1)`
`?(URL)` is an URL (Uniform Resource Locator).	`(url/1)`
`?(P)` is a list of `terms`.	`(list/2)`

Usage 2: `coordinating_group(+Entity, +Key_Project, ?(URL), ?(G))`

– *Description:* When `Entity` is `'group'`, `G` is the list of groups which coordinate the project `Key_Project`. Specifically, if `URL` is non ground, `G` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.

– *Call and exit should be compatible with:*

`+Entity` and `group` unify.	`(= /2)`
`+Key_Project` is an atom which represents a database key.	`(key_db_atm/1)`
`?(URL)` is an URL (Uniform Resource Locator).	`(url/1)`
`?(G)` is a list of `terms`.	`(list/2)`

contact_person_group/4: PREDICATE

Usage 1: `contact_person_group(+Entity, +Key_Group, ?(URL), ?(P))`

– *Description:* When `Entity` is `'person'`, `P` is the list of contact people of the group `Key_Group`. Specifically, if `URL` is non ground, `P` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.

– *Call and exit should be compatible with:*

`+Entity` and `person` unify. (`= /2`)

`+Key_Group` is an atom which represents a database key. (`key_db_atm/1`)

`?(URL)` is an URL (Uniform Resource Locator). (`url/1`)

`?(P)` is a list of `terms`. (`list/2`)

Usage 2: `contact_person_group(+Entity, +Key_Person, ?(URL), ?(G))`

– *Description:* When `Entity` is `'group'`, `G` is the list of groups whose contact person is `Key_Person`. Specifically, if `URL` is non ground, `G` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.

– *Call and exit should be compatible with:*

`+Entity` and `group` unify. (`= /2`)

`+Key_Person` is an atom which represents a database key. (`key_db_atm/1`)

`?(URL)` is an URL (Uniform Resource Locator). (`url/1`)

`?(G)` is a list of `terms`. (`list/2`)

related_group/4: PREDICATE

Usage 1: `related_group(+Entity, +Key_Group, ?(URL), ?(S))`

– *Description:* When `Entity` is `'software'`, `S` is the list of software associated to the group `Key_Group`. Specifically, if `URL` is non ground, `S` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.

– *Call and exit should be compatible with:*

`+Entity` and `software` unify. (`= /2`)

`+Key_Group` is an atom which represents a database key. (`key_db_atm/1`)

`?(URL)` is an URL (Uniform Resource Locator). (`url/1`)

`?(S)` is a list of `terms`. (`list/2`)

Usage 2: `related_group(+Entity, +Key_Software, ?(URL), ?(G))`

– *Description:* When `Entity` is `'group'`, `G` is the list of groups which is related with the software `Key_Software`. Specifically, if `URL` is non ground, `G` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.

– *Call and exit should be compatible with:*

`+Entity` and `group` unify. (`= /2`)

`+Key_Software` is an atom which represents a database key. (`key_db_atm/1`)

`?(URL)` is an URL (Uniform Resource Locator). (`url/1`)

`?(G)` is a list of `terms`. (`list/2`)

contact_person_soft/4: PREDICATE

Usage 1: `contact_person_soft(+Entity, +Key_Person, ?(URL), ?(S))`

– *Description:* When `Entity` is `'software'`, `S` is the list of software whose contact person is `Key_Person`. Specifically, if `URL` is non ground, `S` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.

– *Call and exit should be compatible with:*

`+Entity` and `software` unify. (`= /2`)

`+Key_Person` is an atom which represents a database key. (`key_db_atm/1`)

`?(URL)` is an URL (Uniform Resource Locator). (`url/1`)

`?(S)` is a list of `terms`. (`list/2`)

Usage 2: `contact_person_soft(+Entity, +Key_Software, ?(URL), ?(P))`

- *Description:* When `Entity` is `'person'`, `P` is the list of contact people of the software `Key_Software`. Specifically, if `URL` is non ground, `P` is a list of *Key-Name* pairs. On the contrary, it is a list of *URL-Name* pairs.
- *Call and exit should be compatible with:*

  `+Entity` and `person` unify. (`= /2`)

  `+Key_Software` is an atom which represents a database key. (`key_db_atm/1`)

  `?(URL)` is an URL (Uniform Resource Locator). (`url/1`)

  `?(P)` is a list of `terms`. (`list/2`)

20.3 Documentation on multifiles (`queries_db`)

$is_persistent/2: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

20.4 Version/Change Log (`queries_db`)

Version 0.1#16 (2003/12/5, 12:51:18 CET)

Started automatic documentation (Jorge Navas)

21 Database operations

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#17 (2003/12/5, 12:51:34 CET)

This module implements a set of typical operations over the database. Specifically, it allows defining relationships among the database entities.

21.1 Usage and interface (`add_relations`)

- **Library usage:**
  `:- use_module(library(add_relations)).`
- **Exports:**
  - *Predicates:*
    `add_member_of/3`, `add_contact_person_group/3`, `add_coordinating_group/3`, `add_related_group/3`, `add_contact_person_soft/3`, `add_working/3`.
  - *Multifiles:*
    `$is_persistent/2`.
- **Other modules used:**
  - *Application modules:*
    `../../database/database`.
  - *System library modules:*
    `persdb/persdbrt`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

21.2 Documentation on exports (`add_relations`)

add_member_of/3: PREDICATE

Defines a `'member_of'` relationship between a key `Key` and a list of keys `Keys`. The `'member_of'` relationship is defined by the `member_of/2` predicate in Chapter 19 [Database persistent predicates], page 105.

Usage 1: `add_member_of(+Entity, +Key, +Keys)`

- *Description:* When `Entity` is `'groups'`, `Key` is a group's key, and each element of `Keys` is a person's key.
- *Call and exit should be compatible with:*
  `+Entity` and `groups` unify. (`= /2`)
  `+Key` is an atom which represents a database key. (`key_db_atm/1`)
  `+Keys` is a list of `key_db_ints`. (`list/2`)

Usage 2: `add_member_of(+Entity, +Key, +Keys)`

- *Description:* When `Entity` is `'people'`, `Key` is a person's key, and each element of `Keys` is a group's key.

– *Call and exit should be compatible with:*

`+Entity` and `people` unify. (`= /2`)

`+Key` is an atom which represents a database key. (`key_db_atm/1`)

`+Keys` is a list of `key_db_ints`. (`list/2`)

add_contact_person_group/3: PREDICATE

Defines a `'contact_person_group'` relationship between a key `Key` and a list of keys `Keys`. The `'contact_person_group'` relationship is defined by the `contact_person_group/2` predicate in Chapter 19 [Database persistent predicates], page 105.

Usage 1: `add_contact_person_group(+Entity, +Key, +Keys)`

– *Description:* When `Entity` is `'groups'`, `Key` is a group's key, and each element of `Keys` is a person's key.

– *Call and exit should be compatible with:*

`+Entity` and `groups` unify. (`= /2`)

`+Key` is an atom which represents a database key. (`key_db_atm/1`)

`+Keys` is a list of `key_db_ints`. (`list/2`)

Usage 2: `add_contact_person_group(+Entity, +Key, +Keys)`

– *Description:* When `Entity` is `'people'`, `Key` is a person's key, and each element of `Keys` is a group's key.

– *Call and exit should be compatible with:*

`+Entity` and `people` unify. (`= /2`)

`+Key` is an atom which represents a database key. (`key_db_atm/1`)

`+Keys` is a list of `key_db_ints`. (`list/2`)

add_coordinating_group/3: PREDICATE

Defines a `'coordinating_group'` relationship between a key `Key` and a list of keys `Keys`. The `'coordinating_group'` relationship is defined by the `coordinating_group/2` predicate in Chapter 19 [Database persistent predicates], page 105.

Usage 1: `add_coordinating_group(+Entity, +Key, +Keys)`

– *Description:* When `Entity` is `'groups'`, `Key` is a group's key, and each element of `Keys` is a key of a project.

– *Call and exit should be compatible with:*

`+Entity` and `groups` unify. (`= /2`)

`+Key` is an atom which represents a database key. (`key_db_atm/1`)

`+Keys` is a list of `key_db_ints`. (`list/2`)

Usage 2: `add_coordinating_group(+Entity, +Key, +Keys)`

– *Description:* When `Entity` is `'projects'`, `Key` is a key of a project, and each element of `Keys` is a group's key.

– *Call and exit should be compatible with:*

`+Entity` and `projects` unify. (`= /2`)

`+Key` is an atom which represents a database key. (`key_db_atm/1`)

`+Keys` is a list of `key_db_ints`. (`list/2`)

add_related_group/3: PREDICATE

Defines a `'related_group'` relationship between a key `Key` and a list of keys `Keys`. The `'related_group'` relationship is defined by the `related_group/2` predicate in Chapter 19 [Database persistent predicates], page 105.

Usage 1: `add_related_group(+Entity, +Key, +Keys)`

- *Description:* When `Entity` is `'groups'`, `Key` is a group's key, and each element of `Keys` is a key of software.
- *Call and exit should be compatible with:*

`+Entity` and `groups` unify.	(`= /2`)
`+Key` is an atom which represents a database key.	(`key_db_atm/1`)
`+Keys` is a list of `key_db_ints`.	(`list/2`)

Usage 2: `add_related_group(+Entity, +Key, +Keys)`

- *Description:* When `Entity` is `'software'`, `Key` is a key of software, and each element of `Keys` is a group's key.
- *Call and exit should be compatible with:*

`+Entity` and `software` unify.	(`= /2`)
`+Key` is an atom which represents a database key.	(`key_db_atm/1`)
`+Keys` is a list of `key_db_ints`.	(`list/2`)

add_contact_person_soft/3: PREDICATE

Defines a `'contact_person_soft'` relationship between a key `Key` and a list of keys `Keys`. The `'contact_person_soft'` relationship is defined by the `contact_person_soft/2` predicate in Chapter 19 [Database persistent predicates], page 105.

Usage 1: `add_contact_person_soft(+Entity, +Key, +Keys)`

- *Description:* When `Entity` is `'people'`, `Key` is a person's key, and each element of `Keys` is a key of software.
- *Call and exit should be compatible with:*

`+Entity` and `people` unify.	(`= /2`)
`+Key` is an atom which represents a database key.	(`key_db_atm/1`)
`+Keys` is a list of `key_db_ints`.	(`list/2`)

Usage 2: `add_contact_person_soft(+Entity, +Key, +Keys)`

- *Description:* When `Entity` is `'software'`, `Key` is a key of software, and each element of `Keys` is a person's key.
- *Call and exit should be compatible with:*

`+Entity` and `software` unify.	(`= /2`)
`+Key` is an atom which represents a database key.	(`key_db_atm/1`)
`+Keys` is a list of `key_db_ints`.	(`list/2`)

add_working/3: PREDICATE

Defines a `'working'` relationship between a key `Key` and a list of keys `Keys`. The `'working'` relationship is defined by the `working/2` predicate in Chapter 19 [Database persistent predicates], page 105.

Usage 1: `add_working(+Entity, +Key, +Keys)`

– *Description:* When `Entity` is `'groups'`, `Key` is a group's key, and each element of `Keys` is a key of area.
– *Call and exit should be compatible with:*

`+Entity` and `groups` unify. (`= /2`)

`+Key` is an atom which represents a database key. (`key_db_atm/1`)

`+Keys` is a list of `key_db_ints`. (`list/2`)

Usage 2: `add_working(+Entity, +Key, +Keys)`

– *Description:* When `Entity` is `'areas'`, `Key` is a key of an area, and each element of `Keys` is a group's key.
– *Call and exit should be compatible with:*

`+Entity` and `areas` unify. (`= /2`)

`+Key` is an atom which represents a database key. (`key_db_atm/1`)

`+Keys` is a list of `key_db_ints`. (`list/2`)

21.3 Documentation on multifiles (`add_relations`)

$is_persistent/2: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

21.4 Version/Change Log (`add_relations`)

Version 0.1#17 (2003/12/5, 12:51:34 CET)

Started automatic documentation (Jorge Navas)

22 Database types

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#18 (2003/12/5, 12:51:57 CET)

This module consists of a set of basic types used by the predicates defined in Chapter 19 [Database persistent predicates], page 105.

22.1 Usage and interface (types)

- **Library usage:**
  `:- use_module(library(types)).`
- **Exports:**
  - *Regular Types:*
    `group_struct/1`, `person_struct/1`, `project_struct/1`, `software_struct/1`, `event_struct/1`, `area_struct/1`, `group_red/1`, `person_red/1`, `project_red/1`, `software_red/1`, `event_red/1`, `area_red/1`.
- **Other modules used:**
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

22.2 Documentation on exports (types)

group_struct/1: REGTYPE

`group_struct(Group_Struct)`

This type contains data about a group. It is defined as:

```
group_struct(group(Date,Key,Password,Inst_a,Inst_f,WWW_inst,
                   Group_a,Group_f,WWW_group,Category,Description,
                   Address,Zip,State,City,Country)).
```

The arguments shown above mean:

- `Date` is the date when the group was registered. It is defined by the system.
- `Key` is a unique identifier. It is defined by the system.
- `Password` is an access key.
- `Inst_a` is the institution's abbreviated name to which the group belongs to.
- `Inst_f` is the institution's complete name to which the group belongs to.
- `WWW_inst` is the institution's Web address to which the group belongs to.
- `Group_a` is the abbreviated name.
- `Group_f` is the complete name.
- `WWW_group` is the Web address.
- `Category` is the category.
- `Description` is a brief description.

- `Address` is the address.
- `Zip` is the zip code.
- `State` is the state (it is optional).
- `City` is the city.
- `Country` is the country.

Usage: `group_struct(Group_Struct)`

– *Description:* `Group_Struct` is a term that contains data about a group.

person_struct/1: REGTYPE

`person_struct(Person_Struct)`

This type contains data about a person. It is defined as:

```
person_struct(person(Date,Key,Password,Last_Name,First_Name,Title,
                     WWW,Email,Phone,Fax,Description,Address,Zip,
                     State,City,Country)).
```

The arguments shown above mean:

- `Date` is the date when the person was registered. It is defined by the system.
- `Key` is a unique identifier. It is defined by the system.
- `Password` is an access key.
- `Last_Name` is the last name.
- `First_Name` is the first name.
- `Title` is the degree.
- `WWW` is the Web address.
- `Email` is the email.
- `Phone` is the telephone number.
- `Fax` is the fax number.
- `Description` is a brief description.
- `Address` is the address.
- `Zip` is the zip code.
- `State` is the state (it is optional).
- `City` is the city.
- `Country` is the country.

Usage: `person_struct(Person_Struct)`

– *Description:* `Person_Struct` is a term that contains data about a person.

project_struct/1: REGTYPE

`project_struct(Project_Struct)`

This type contains data about a project. It is defined as:

```
project_struct(project(Date,Key,Password,Title_a,Title_f,WWW,
                       Category,Description,Comment)).
```

The arguments shown above mean:

- `Date` is the date when the project was registered. It is defined by the system.
- `Key` is a unique identifier.It is defined by the system.
- `Password` is an access key.
- `Title_a` is the abbreviated title.
- `Title_f` is the complete title.
- `WWW` is the Web address.
- `Category` is the category.
- `Description` is a brief description.
- `Comment` is an additional comment.

Usage: `project_struct(Project_Struct)`

– *Description:* `Project_Struct` is a term that contains data about a project.

software_struct/1: REGTYPE

`software_struct(Software_Struct)`

This type represents data about a software. It is defined as:

```
software_struct(software(Date,Key,Password,Name_a,Name_f,WWW,FTP,
                         Platforms,Availability,Description)).
```

The arguments shown above mean:

- `Date` is the date when the project was registered. It is defined by the system.
- `Key` is a unique identifier. It is defined by the system.
- `Password` is an access key.
- `Name_a` is the abbreviated name.
- `Name_f` is the complete name.
- `WWW` is the Web address.
- `FTP` is the FTP address.
- `Platforms` are the platforms supported.
- `Availability` is the type of license.
- `Description` is a brief description.

Usage: `software_struct(Software_Struct)`

– *Description:* `Software_Struct` is a term that contains data about a software.

event_struct/1: REGTYPE

`event_struct(Event_Struct)`

This type represents data about an event. It is defined as:

```
event_struct(event(Date_first,Key,Password,Title,Date_last,Deadline,
                   Category,City,Country,Comment,Person_Contact,
                   Email)).
```

The arguments shown above mean:

- `Date_first` is the start date.
- `Key` is a unique identifier. It is defined by the system.

- `Password` is an access key.
- `Date_last` is the end date.
- `Deadline` is the deadline date.
- `Title` is the title.
- `Category` is the category.
- `City` is the city where the event is held.
- `Country` is the country where the event is held.
- `Comment` is an additional comment.
- `Person_Contact` is the contact person.
- `Email` is the contact person's email.

Usage: `event_struct(Event_Struct)`

– *Description:* `Event_Struct` is a term that contains data about an event.

area_struct/1: REGTYPE

`area_struct(Area_Struct)`

This type contains data about an area. It is defined as:

```
area_struct(area(Name_a,Name_f,Key,Date,Description,Password)).
```

The arguments shown above mean:

- `Name_a` is the abbreviated name.
- `Name_f` is the complete name.
- `Key` is a unique identifier. It is defined by the system.
- `Date` is the date when the area was registered. It is defined by the system.
- `Description` is a brief description.
- `Password` is an access key.

Usage: `area_struct(Area_Struct)`

– *Description:* `Area_Struct` is a term that contains data about an area.

group_red/1: REGTYPE

Usage: `group_red(G)`

– *Description:* `G` contains reduced information (key and name) about a group

person_red/1: REGTYPE

Usage: `person_red(Per)`

– *Description:* `Per` contains reduced information (key and name) about a person.

project_red/1: REGTYPE

Usage: `project_red(Pro)`

– *Description:* `Pro` contains reduced information (key and name) about a project.

software_red/1: REGTYPE

Usage: `software_red(S)`

– *Description:* `S` contains reduced information (key and name) about a software.

event_red/1: REGTYPE

Usage: `event_red(E)`

– *Description:* `E` contains reduced information (key and name) about an event.

area_red/1: REGTYPE

Usage: `area_red(A)`

– *Description:* `A` contains reduced information (key and name) about an area.

22.3 Version/Change Log (`types`)

Version 0.1#18 (2003/12/5, 12:51:57 CET)

Started automatic documentation (Jorge Navas)

PART IV - Ciao Prolog extensions

This part includes a set of libraries that are an extension of the Ciao libraries.

23 The PiLLoW Web programming library extension

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#19 (2003/12/5, 12:52:31 CET)

This library is an extension of the `PiLLoW` library, defining new functionalities, related with parsing, manipulating and generating HTML and XML structured documents and data; producing HTML forms; writing form handlers and CGI-scripts and HTML/ XML templates.

23.1 Usage and interface (`pillow_ext`)

- **Library usage:**
  `:- use_module(library(pillow_ext)).`
- **Exports:**
  - *Predicates:*
    `get_form_values/3`, `form_value_empty/1`, `form_value_default/2`, `url_query/3`, `output_html/2`, `get_xmlfile_input/2`, `get_xmlfile_value/3`, `get_xmlfile_node/3`, `get_xmlfile_nodes/3`, `xml_html/1`.
  - *Regular Types:*
    `xml_file/1`, `html_term/1`, `xml_tree/1`, `output_template/1`, `dic/1`, `url/1`.
- **Other modules used:**
  - *Application modules:*
    `./http_aux`, `../strings_ext/strings_ext`, `../../settings`.
  - *System library modules:*
    `pillow/http`, `pillow/html`, `file_utils`, `terms`, `aggregates`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

23.2 Documentation on exports (`pillow_ext`)

get_form_values/3: PREDICATE

Usage: `get_form_values(+Dict, +Var, ?(Values))`

- *Description:* Provides all the `Values` of the variable `Var` from the dictionary `Dict`.
- *Call and exit should be compatible with:*
  `+Dict` is a dictionary of *'attribute-value'* pairs. (`dic/1`)
  `+Var` is an attribute. (`html_attr/1`)
  `?(Values)` is a list of `html_val`s. (`list/2`)

form_value_empty/1: PREDICATE

Usage: `form_value_empty(?(Val))`

- *Description:* If the value `Val` of a variable belonging to a dictionary is empty, this value unifies with `'$empty'`.

– *Call and exit should be compatible with:*
`?(Val)` and `$empty` unify. (`= /2`)

form_value_default/2: PREDICATE

Usage: `form_value_default(?(Val1), ?(Val2))`

– *Description:* The value `Val1` unifies with `Val2`. If `Val1` is empty, `Val2` unify with ‘ ’.

url_query/3: PREDICATE

Usage: `url_query(+URL, +Dict, -URLArgs)`

– *Description:* Provides a URL by translating a dictionary `Dict` of parameter values into a string `URLArgs` for appending to a URL pointing to a `URL`.

– *Call and exit should be compatible with:*
`+URL` is an URL (Uniform Resource Locator). (`url/1`)
`+Dict` is a dictionary of *‘attribute-value’* pairs. (`dic/1`)
`-URLArgs` is an URL (Uniform Resource Locator). (`url/1`)

output_html/2: PREDICATE

Usage: `output_html(+Template, ?(Dict))`

– *Description:* Outputs an `HTML` template with the content of the dictionary `Dic`. Reads the contents of the `Template` file, interpreting it as a HTML template, and obtains an structure of HTML terms, which includes variables, unifying them with the values of the dictionary `Dict`.

– *Call and exit should be compatible with:*
`+Template` is an HTML template. (`html_template/1`)
`?(Dict)` is a dictionary of *‘attribute-value’* pairs. (`dic/1`)

get_xmlfile_input/2: PREDICATE

Usage: `get_xmlfile_input(+XMLString, -Input)`

– *Description:* `Input` is the term which represents the XML code `XMLString`.

– *Call and exit should be compatible with:*
`+XMLString` is a string that represents an XML file. (`xml_string/1`)
`-Input` is a term that represents an XML file. (`xml_input/1`)

get_xmlfile_value/3: PREDICATE

Usage: `get_xmlfile_value(+XMLTree, +Node, ?(Val))`

– *Description:* `Val` is the value of the node `Node` belonging to the tree `XMLTree`. In case that `Val` is empty, it fails.

– *Call and exit should be compatible with:*
`+XMLTree` is an XML file represented as a tree. (`xml_tree/1`)
`+Node` is the name of an XML file node represented as a tree. (`xml_node/1`)
`?(Val)` is the value of an XML file node. (`xml_val/1`)

get_xmlfile_node/3: PREDICATE

Usage: `get_xmlfile_node(+XMLTree, +Node, ?(SubTree))`

- *Description:* `SubTree` is the content of the node `Node` belonging to the tree `XMLTree`. `Node` has not children nodes.
- *Call and exit should be compatible with:*

  `+XMLTree` is an XML file represented as a tree. (`xml_tree/1`)

  `+Node` is the name of an XML file node represented as a tree. (`xml_node/1`)

  `?(SubTree)` is an XML file represented as a tree. (`xml_tree/1`)

get_xmlfile_nodes/3: PREDICATE

Usage: `get_xmlfile_nodes(+XMLTree, +Node, ?(SubTree))`

- *Description:* `SubTree` is the content of the node `Node` belonging to the tree `XMLTree`. `Node` can have children nodes.
- *Call and exit should be compatible with:*

  `+XMLTree` is an XML file represented as a tree. (`xml_tree/1`)

  `+Node` is the name of an XML file node represented as a tree. (`xml_node/1`)

  `?(SubTree)` is an XML file represented as a tree. (`xml_tree/1`)

xml_html/1: PREDICATE

Usage: `xml_html(+XMLFile)`

- *Description:* Translates an XML file `XMLFile` into HTML and outputs it.
- *Call and exit should be compatible with:*

  `+XMLFile` is the name of an XML file. (`xml_file/1`)

xml_file/1: REGTYPE

Usage: `xml_file(F)`

- *Description:* `F` is the name of an XML file.

html_term/1: REGTYPE

Usage: `html_term(T)`

- *Description:* `T` is a `PiLLoW` HTML term.

xml_tree/1: REGTYPE

Usage: `xml_tree(X)`

- *Description:* `X` is an XML file represented as a tree.

output_template/1: REGTYPE

Usage: `output_template(Output)`

- *Description:* `Output` is a term that defines the name of an HTML template to be shown, and a dictionary with values for the variables defined in that template.

dic/1: REGTYPE

Usage: `dic(Dic)`

– *Description:* `Dic` is a dictionary of *'attribute-value'* pairs.

url/1: REGTYPE

Usage: `url(U)`

– *Description:* `U` is an URL (Uniform Resource Locator).

23.3 Documentation on internals (`pillow_ext`)

html_attr/1: REGTYPE

Usage: `html_attr(A)`

– *Description:* `A` is an attribute.

html_val/1: REGTYPE

Usage: `html_val(Value)`

– *Description:* `Value` is a value.

html_template/1: REGTYPE

Usage: `html_template(T)`

– *Description:* `T` is an HTML template.

xml_string/1: REGTYPE

Usage: `xml_string(S)`

– *Description:* `S` is a string that represents an XML file.

xml_type/1: REGTYPE

Usage: `xml_type(T)`

– *Description:* `T` is a type of XML file. The types can be:

- *register*
- *modify*
- *unregister*

xml_attr/1: REGTYPE

Usage: `xml_attr(A)`

– *Description:* `A` is a attribute of an XML file element.

xml_input/1: REGTYPE

Usage: `xml_input(I)`

– *Description:* `I` is a term that represents an XML file.

xml_node/1: REGTYPE

Usage: `xml_node(N)`

– *Description:* `N` is the name of an XML file node represented as a tree.

xml_val/1: REGTYPE

Usage: `xml_val(Value)`

– *Description:* `Value` is the value of an XML file node.

23.4 Version/Change Log (`pillow_ext`)

Version 0.1#19 (2003/12/5, 12:52:31 CET)

Started automatic documentation (Jorge Navas)

24 HTTP protocol basic predicates

Version: 0.1#27 (2003/12/5, 13:14:2 CET)
Version of last change: 0.1#20 (2003/12/5, 12:52:46 CET)

This module implements basic predicates related to the HTTP protocol, which allows retrieving data from HTTP servers.

24.1 Usage and interface (http_aux)

- **Library usage:**
  `:- use_module(library(http_aux)).`
- **Exports:**
  - *Predicates:*
    `http_response/3`.
- **Other modules used:**
  - *System library modules:*
    `pillow/pillow_aux`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

24.2 Documentation on exports (http_aux)

http_response/3: PREDICATE

Usage: `http_response(?(Response), +ResponseChars, +RestResponseChars)`

- *Description:* `Response` is the result of processing the server response `ResponseChars` and the rest of this response `RestResponseChars`.
- *Call and exit should be compatible with:*
  `?(Response)` is an HTTP protocol header. (`http_head/1`)
  `+ResponseChars` is a string (a list of character codes). (`string/1`)
  `+RestResponseChars` is a string (a list of character codes). (`string/1`)

24.3 Documentation on internals (http_aux)

http_head/1: REGTYPE

Usage: `http_head(H)`

- *Description:* `H` is an HTTP protocol header.

24.4 Version/Change Log (http_aux)

Version 0.1#20 (2003/12/5, 12:52:46 CET)
Started automatic documentation (Jorge Navas)

25 List processing extension

Version: 0.1#27 (2003/12/5, 13:14:2 CET)
Version of last change: 0.1#21 (2003/12/5, 12:53:17 CET)
This library is an extension of the `list` library, defining new functionalities.

25.1 Usage and interface (`lists_ext`)

- **Library usage:**
  `:- use_module(library(lists_ext)).`
- **Exports:**
  - *Predicates:*
    `delete_repeated/2`.
- **Other modules used:**
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

25.2 Documentation on exports (`lists_ext`)

delete_repeated/2: PREDICATE

Usage: `delete_repeated(+L1, ?(L2))`

- *Description:* `L2` is `L1` without repeated elements.
- *Call and exit should be compatible with:*
  `+L1` is a list. (`list/1`)
  `?(L2)` is a list. (`list/1`)

25.3 Version/Change Log (`lists_ext`)

Version 0.1#21 (2003/12/5, 12:53:17 CET)
Started automatic documentation (Jorge Navas)

26 String processing extension

Version: 0.1#27 (2003/12/5, 13:14:2 CET)
Version of last change: 0.1#22 (2003/12/5, 12:53:43 CET)
This library is an extension of the `string` library, defining new functionalities.

26.1 Usage and interface (`strings_ext`)

- **Library usage:**
  `:- use_module(library(strings_ext)).`
- **Exports:**
  - *Predicates:*
    `removed_delimitator/2`, `concat_line/2`, `value_default/2`.
- **Other modules used:**
  - *System library modules:*
    `lists`, `terms`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

26.2 Documentation on exports (`strings_ext`)

removed_delimitator/2: PREDICATE
Usage: `removed_delimitator(+String1, ?(String2))`
- *Description:* `String2` unify with `String1` but without delimitators (a positive number of space (32), tab (9), newline (10) or return (13) characters) neither at the beginning nor at the end of the string.
- *Call and exit should be compatible with:*
  `+String1` is a string (a list of character codes). (`string/1`)
  `?(String2)` is a string (a list of character codes). (`string/1`)

concat_line/2: PREDICATE
Usage: `concat_line(LL, L)`
- *Description:* `L` is the concatenation of all the lists in `LL`, separated by `' '`.
- *Call and exit should be compatible with:*
  `LL` is a list of `lists`. (`list/2`)
  `L` is a list. (`list/1`)

value_default/2: PREDICATE
Usage: `value_default(?(Var), ?(Val))`
- *Description:* `Var` unifies with `Val`. If `Var` is empty, `Val` unify with `' '`.

26.3 Version/Change Log (`strings_ext`)

Version 0.1#22 (2003/12/5, 12:53:43 CET)
Started automatic documentation (Jorge Navas)

27 Persistent counters processing

Version: 0.1#27 (2003/12/5, 13:14:2 CET)

Version of last change: 0.1#23 (2003/12/5, 12:54:4 CET)

This library is a redefinition of the `counter` library, where the counters are stored by *persistent predicates*.

27.1 Usage and interface (`counters_ext`)

- **Library usage:**
  `:- use_module(library(counters_ext)).`
- **Exports:**
  - *Predicates:*
    `egetcounter/2`, `egetcircularcounter/2`, `esetcounter/2`.
  - *Multifiles:*
    `$is_persistent/2`.
- **Other modules used:**
  - *Application modules:*
    `../../database/database`.
  - *System library modules:*
    `persdb/persdbrt`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

27.2 Documentation on exports (`counters_ext`)

egetcounter/2: PREDICATE

Usage: `egetcounter(?(Name), ?(Val))`

- *Description:* Returns in `Val` the current value of the `Name` counter, and increases it one unit.
- *Call and exit should be compatible with:*
  `?(Name)` is a atom which represents a counter. (`name/1`)
  `?(Val)` is a atom which represents the value of a counter. (`val/1`)

egetcircularcounter/2: PREDICATE

Usage: `egetcircularcounter(?(Name), ?(Val))`

- *Description:* Returns in `Val` the current value of the `Name` counter, and increases it one unit. When it reaches to the established end, it begins from zero. This end is defined in 10000.

– *Call and exit should be compatible with:*
`?(Name)` is a atom which represents a counter. (`name/1`)
`?(Val)` is a atom which represents the value of a counter. (`val/1`)

esetcounter/2: PREDICATE

Usage: `esetcounter(+Name, +Val)`

– *Description:* Assigns the counter `Name` to the value `Val`.
– *Call and exit should be compatible with:*
`+Name` is a atom which represents a counter. (`name/1`)
`+Val` is a atom which represents the value of a counter. (`val/1`)

27.3 Documentation on multifiles (counters_ext)

$is_persistent/2: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

27.4 Documentation on internals (counters_ext)

counter/2: PREDICATE

The predicate is of type *data*.

Usage: `counter(?(Name), ?(Val))`

– *Call and exit should be compatible with:*
`?(Name)` is a atom which represents a counter. (`name/1`)
`?(Val)` is a atom which represents the value of a counter. (`val/1`)

name/1: REGTYPE

Usage: `name(N)`

– *Description:* `N` is a atom which represents a counter.

val/1: REGTYPE

Usage: `val(V)`

– *Description:* `V` is a atom which represents the value of a counter.

27.5 Version/Change Log (counters_ext)

Version 0.1#23 (2003/12/5, 12:54:4 CET)

Started automatic documentation (Jorge Navas)

PART V - Application libraries

This part includes a set of libraries not defined in Ciao, and that are required in order to the application works correctly.

28 Date processing

Version: 0.1#27 (2003/12/5, 13:14:2 CET)
Version of last change: 0.1#24 (2003/12/5, 12:54:26 CET)
This library defines a set of predicates that allows handling and manipulating dates.

28.1 Usage and interface (date)

- **Library usage:**
  `:- use_module(library(date)).`
- **Exports:**
  - *Predicates:*
    `cod_month/2`, `dec_month/4`, `get_date/4`, `set_date_f0/4`, `set_date_f1/4`, `set_date_f2/2`.
  - *Regular Types:*
    `date/1`.
- **Other modules used:**
  - *System library modules:*
    `system`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

28.2 Documentation on exports (date)

date/1: REGTYPE

Usage: `date(D)`

- *Description:* `D` is a date.

cod_month/2: PREDICATE

Usage: `cod_month(?(Month), ?(Month_Textual))`

- *Description:* `Month_Textual` is the name of the `Month` month.
- *Call and exit should be compatible with:*
  `?(Month)` is a number which represents a month. (`month/1`)
  `?(Month_Textual)` is the name of a month. (`month_textual/1`)

dec_month/4: PREDICATE

Usage: `dec_month(?(Year), +Month, ?(Day), ?(Date))`

- *Description:* `Date` is the date consisting of `Year`, `Month` and `Day` decreased in one month.

– *Call and exit should be compatible with:*

`?(Year)` is a year. (`year/1`)

`+Month` is a number which represents a month. (`month/1`)

`?(Day)` is a day. (`day/1`)

`?(Date)` is a date. (`date/1`)

get_date/4: PREDICATE

Usage: `get_date(+Date, ?(Year), ?(Month), ?(Day))`

– *Description:* Provides the `Year`, `Month` and `Day` of the date `Date` represented in the **year-month-day** format.

– *Call and exit should be compatible with:*

`+Date` is an atom. (`atm/1`)

`?(Year)` is a year. (`year/1`)

`?(Month)` is a number which represents a month. (`month/1`)

`?(Day)` is a day. (`day/1`)

set_date_f0/4: PREDICATE

Usage: `set_date_f0(?(Date), +Year, +Month, +Day)`

– *Description:* Provides the `Date` date in the **year-month-day** format from `Year`, `Month` and `Day`.

– *Call and exit should be compatible with:*

`?(Date)` is an atom. (`atm/1`)

`+Year` is an atom which represents a year. (`year_atm/1`)

`+Month` is the name of a month. (`month_textual/1`)

`+Day` is an atom which represents a day. (`day_atm/1`)

set_date_f1/4: PREDICATE

Usage: `set_date_f1(?(Date), +Year, +Month, +Day)`

– *Description:* Provides the `Date` date in the **year-month-day** format from `Year`,`Month` and `Day`.

– *Call and exit should be compatible with:*

`?(Date)` is an atom. (`atm/1`)

`+Year` is a year. (`year/1`)

`+Month` is a number which represents a month. (`month/1`)

`+Day` is a day. (`day/1`)

set_date_f2/2: PREDICATE

Usage: `set_date_f2(+Time, ?(Time_Format))`

– *Description:* Provides the `Time_Format` date in the **year month day hour:minute:second** format from `Time`.

– *Call and exit should be compatible with:*

`+Time` is an structure which stores the current date and time. (`datime_struct/1`)

`?(Time_Format)` is an atom. (`atm/1`)

28.3 Documentation on internals (`date`)

year/1: REGTYPE

Usage: `year(Y)`

– *Description:* `Y` is a year.

month/1: REGTYPE

Usage: `month(M)`

– *Description:* `M` is a number which represents a month.

day/1: REGTYPE

Usage: `day(D)`

– *Description:* `D` is a day.

month_textual/1: REGTYPE

Usage: `month_textual(M)`

– *Description:* `M` is the name of a month.

year_atm/1: REGTYPE

Usage: `year_atm(Y)`

– *Description:* `Y` is an atom which represents a year.

day_atm/1: REGTYPE

Usage: `day_atm(D)`

– *Description:* `D` is an atom which represents a day.

datime_struct/1: REGTYPE

Usage: `datime_struct(D)`

– *Description:* `D` is an structure which stores the current date and time.

28.4 Version/Change Log (`date`)

Version 0.1#24 (2003/12/5, 12:54:26 CET)

Started automatic documentation (Jorge Navas)

29 Password processing

Version: 0.1#27 (2003/12/5, 13:14:2 CET)
Version of last change: 0.1#25 (2003/12/5, 12:54:45 CET)
This library defines a set of predicates that allows handling and manipulating passwords.

29.1 Usage and interface (password)

- **Library usage:**
  `:- use_module(library(password)).`
- **Exports:**
  - *Predicates:*
    `get_password/1`, `process_password/2`.
- **Other modules used:**
  - *Application modules:*
    `../strings_ext/strings_ext`.
  - *System library modules:*
    `random/random`, `terms`, `dec10_io`.
  - *Internal (engine) modules:*
    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

29.2 Documentation on exports (password)

get_password/1: PREDICATE
Usage: `get_password(-Password)`
- *Call and exit should be compatible with:*
  `-Password` is a eight items long pseudoramdom string. (`password/1`)

process_password/2: PREDICATE
Usage: `process_password(+Password1, ?(Password2))`
- *Description:* `Password1` unifies with `Password2` but without blanks.

29.3 Documentation on internals (password)

password/1: REGTYPE
Usage: `password(P)`
- *Description:* `P` is a eight items long pseudoramdom string.

30 Installing CoLogNetWS

30.1 Installation steps

The following provides the different steps which are necessary to make a complete installation of `CoLogNetWS`.

- Uncompress (using `gunzip`) and unpackage (using `tar -xpf`) the distribution in a suitable directory.
- Edit the `SETTINGS` file and change values to suit your installation.
- Edit the `settings.pl` file and change values to suit your installation.
- In a shell, cd to the CoLogNetWS root and run `make all`. This creates the CGI executables, creates the target directories, and copies several `files` to their destinations in these directories.
- PERMISSIONS:
  - The directory `private` will contain the CGI executables which allow the management of the Web-site. You will probably want to protect it with an `HTTP password`. In this directory you can find the scripts for manipulating password files for `NCSA httpd`.

    Ideally, `private` should not be readable by anyone (but needs to be readable by the HTTP daemon, of course).
  - Check that your system's httpd configuration file allows the CGI executables (all ending in .cgi) located in `cgi-bin` and `private` to be run from a browser. Also, the user under which the httpd is run needs to have write permission in directories `database/database`, `database/counters_ext`, and `lib/security/authentication`.
- You are done! If you want to delete temporary files created during the installation, run `make clean` in the source directory.

30.2 Usage Summary

A very brief summary of usage (you should consult the relevant sections of the manual for a full explanation):

- To consult the database (no modification allowed), see the directory `database/database` and its files.
- To consult the XML files which are stored in the intermediate buffers (no modification allowed), see the directory `lib/exchange/sending` and its subdirectories.
- To consult the log file which stores the errors produced during data exchange with other Web-sites (no modification allowed), see the file `lib/exchange/errors_log/errors_xml.pl`.
- To consult and modify the group and event categories, see the file `lib/util/category.pl`.
- To consult and modify the Web-site URL's, with which it exists a communication, see the file `lib/util/url_websites.pl`.

30.3 Customization

Extensive customization of the appearance of the application Web pages is possible:

- The layout of the pages is determined by the HTML files in the `templates/html` directory. You can change these files (between the body tags) to change the appearance of the HTML pages. The title of each template file explains what the template is used for, and lists the template variables between parenthesis. In these files, the template variables are written

like this `<V>varname</V>` if they appear in normal text or like `_varname`, if they appear as an attribute or attribute value of a tag.

You can change these templates by hand or using an HTML editor. Beware, however, of WYSIWYG HTML editors! Always check that they have not arbitrarily changed attributes, tag ordering, etc.

- The layout of the XML file is determined by the XML files in the `templates/xml` directory. You can change these files (between the body tags) to change the appearance of the XML files. The title of each template file explains what the template is used for, and lists the template variables between parenthesis. In these files, the template variables are written like this `<V>varname</V>` if they appear in normal text or like `_varname`, if they appear as an attribute or attribute value of a tag.

  You can change these templates by hand or using an XML editor. Beware, however, of WYSIWYG XML editors! Always check that they have not arbitrarily changed attributes, tag ordering, etc.

References

[CH01] D. Cabeza and M. Hermenegildo.
Distributed WWW Programming using (Ciao-)Prolog and the PiLLoW Library.
Theory and Practice of Logic Programming, 1(3):251–282, May 2001.

[GCH98] J.M. Gomez, D. Cabeza, and M. Hermenegildo.
persdb: Persistent Database Interface.
Technical Report D3.1.M2-A1 CLIP9/98.0, RADIOWEB Project, December 1998.